2nd International Conference on Formal Structures for Computation and Deduction

FSCD 2017, September 3–9, 2017, Oxford, UK

Edited by

Dale Miller

Editor

Dale Miller
Inria, UMR 7161, LIX, École Polytechnique
Palaiseau
France
dale@lix.polytechnique.edu

ACM Classification 1998
F. Theory of Computation, F.1 Computation by Abstract Devices, F.2 Analysis of Algorithms and Problem Complexity, F.3 Logics and Meanings of Programs, F.4 Mathematical Logic and Formal Languages, D.1 Programming Techniques, D.2.4 Software/Program Verification, D.3 Programming Languages, I.2.3 Deduction and Theorem Proving, I.2.4 Knowledge Representation Formalisms and Methods, E.1 Data Structures

ISBN 978-3-95977-047-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-047-7.

Publication date
September, 2017

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

Digital Object Identifier: 10.4230/LIPIcs.FSCD.2017.0

ISBN 978-3-95977-047-7 **ISSN 1868-8969** **http://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

Preface
Dale Miller 0:vii–0:viii

Invited Papers

Type Systems for the Relational Verification of Higher Order Programs
Marco Gaboardi 1:1–1:1

Uniform Resource Analysis by Rewriting: Strenghts and Weaknesses
Georg Moser 2:1–2:10

Brzozowski Goes Concurrent – A Kleene Theorem for Pomset Languages
Alexandra Silva 3:1–3:1

Quantitative Semantics for Probabilistic Programming
Christine Tasson 4:1–4:1

Regular Papers

Displayed Categories
Benedikt Ahrens and Peter LeFanu Lumsdaine 5:1–5:16

The Confluent Terminating Context-Free Substitutive Rewriting System for the lambda-Calculus with Surjective Pairing and Terminal Type
Yohji Akama 6:1–6:19

Improving Rewriting Induction Approach for Proving Ground Confluence
Takahito Aoto, Yoshihito Toyama, and Yuta Kimura 7:1–7:18

Böhm Reduction in Infinitary Term Graph Rewriting Systems
Patrick Bahr 8:1–8:20

Optimality and the Linear Substitution Calculus
Pablo Barenbaum and Eduardo Bonelli 9:1–9:16

Generalized Refocusing: From Hybrid Strategies to Abstract Machines
Małgorzata Biernacka, Witold Charatonik, and Klara Zielińska 10:1–10:17

Nested Multisets, Hereditary Multisets, and Syntactic Ordinals in Isabelle/HOL
Jasmin Christian Blanchette, Mathias Fleury, and Dmitriy Traytel 11:1–11:18

Observably Deterministic Concurrent Strategies and Intensional Full Abstraction for Parallel-or
Simon Castellan, Pierre Clairambault, and Glynn Winskel 12:1–12:16

There Is Only One Notion of Differentiation
J. Robin B. Cockett and Jean-Simon Lemay 13:1–13:21

Confluence of an Extension of Combinatory Logic by Boolean Constants
Łukasz Czajka 14:1–14:16

2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017).
Editor: Dale Miller
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Complexity of Principal Inhabitation
Andrej Dudenhefner and Jakob Rehof 15:1–15:14

List Objects with Algebraic Structure
Marcelo Fiore and Philip Saville 16:1–16:18

Is the Optimal Implementation Inefficient? Elementarily Not
Stefano Guerrini and Marco Solieri 17:1–17:16

Continuation Passing Style for Effect Handlers
Daniel Hillerström, Sam Lindley, Robert Atkey, and K. C. Sivaramakrishnan 18:1–18:19

Infinite Runs in Abstract Completion
Nao Hirokawa, Aart Middeldorp, Christian Sternagel, and Sarah Winkler 19:1–19:16

Refutation of Sallé's Longstanding Conjecture
Benedetto Intrigila, Giulio Manzonetto, and Andrew Polonsky 20:1–20:18

Relating System F and $\lambda 2$: A Case Study in Coq, Abella and Beluga
Jonas Kaiser, Brigitte Pientka, and Gert Smolka 21:1–21:19

A Polynomial-Time Algorithm for the Lambek Calculus with Brackets of Bounded Order
Max Kanovich, Stepan Kuznetsov, Glyn Morrill, and Andre Scedrov 22:1–22:17

Polynomial Running Times for Polynomial-Time Oracle Machines
Akitoshi Kawamura and Florian Steinberg 23:1–23:18

Types as Resources for Classical Natural Deduction
Delia Kesner and Pierre Vial 24:1–24:17

A Fibrational Framework for Substructural and Modal Logics
Daniel R. Licata, Michael Shulman, and Mitchell Riley 25:1–25:22

Arrays and References in Resource Aware ML
Benjamin Lichtman and Jan Hoffmann 26:1–26:20

Negative Translations and Normal Modality
Tadeusz Litak, Miriam Polzer, and Ulrich Rabenstein 27:1–27:18

Models of Type Theory Based on Moore Paths
Ian Orton and Andrew M. Pitts 28:1–28:16

On Dinaturality, Typability and $\beta\eta$-Stable Models
Paolo Pistone 29:1–29:17

A Curry-Howard Approach to Church's Synthesis
Pierre Pradic and Colin Riba 30:1–30:16

Combinatorial Flows and Their Normalisation
Lutz Straßburger 31:1–31:17

Streett Automata Model Checking of Higher-Order Recursion Schemes
Ryota Suzuki, Koichi Fujima, Naoki Kobayashi, and Takeshi Tsukada 32:1–32:18

A Sequent Calculus for a Semi-Associative Law
Noam Zeilberger 33:1–33:16

Preface

The Second International Conference on Formal Structures for Computation and Deduction (FSCD 2017) was held 3–9 September 2017 in the Mathematical Institute at the University of Oxford, UK. This volume contains the proceedings of that meeting.

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

FSCD 2017 has received 77 submissions with contributing authors from 23 countries. The program committee consisted of 31 members from 12 countries. Each submitted paper was reviewed by at least 3 members of the program committee with the help of 105 external reviewers. The reviewing process, which included a rebuttal phase, took place over a period of 8 weeks. A total of 29 regular papers were accepted for publication and are included in these proceedings. The FSCD program also featured four invited talks given by Marco Gaboardi (State University of New York at Buffalo), Georg Moser (University of Innsbruck), Alexandra Silva (University College London), and Christine Tasson (Université Paris Diderot).

FSCD 2017 was co-located with the 22nd ACM SIGPLAN International Conference on Functional Programming (ICFP 2017). In additional to the main program, the following 11 FSCD-associated workshops were held the day proceeding and the two days following the main conference.

- Trends in Linear Logic and Applications
- 31st International Workshop on Unification
- Trends in Mechanised Security Proofs, COST Action CA15123 EUTypes Workshop
- Third Workshop on Higher-Dimensional Rewriting and Applications
- Third Workshop on Homotopy Type Theory and Univalent Foundations
- First Annual Workshop on String Diagrams in Computation, Logic, and Physics
- Fourth Meeting on Structures and Deduction
- Sixth International Workshop on Confluence
- Twelfth International Workshop on Logical Frameworks and Meta-Languages – Theory and Practice
- Fourth International Workshop on Rewriting Techniques for Program Transformation and Evaluation
- IFIP Working Group 1.6: Rewriting

This volume of FSCD 2017 is being published in the LIPIcs series under a Creative Common license: online access is free to all and authors retaining rights over their contributions. We thank in particular Marc Herbstritt from the Leibniz Center for Informatics at Schloss Dagstuhl for his support during the production of these proceedings.

Many people helped to make FSCD 2017 into a successful meeting. On behalf of the program committee, I thank the many authors of submitted papers for considering FSCD as a venue for their work and the invited speakers who have agreed to speak at this meeting. The program committee and the external reviewers deserve a big thanks for their carefully reviewing and evaluating of the submitted papers (the members of the program committee

and the list of external reviewers can be found in the following pages). The many associated workshops made a big contribution to the lively scientific atmosphere of this meeting and I thank the workshop organizers for their efforts to bring their meetings to Oxford. The EasyChair conference management system once again proved its usefulness at managing all the phases of putting together a program for this meeting.

The hard work of Sam Staton, the Conference Chair for FSCD 2017, is responsible for the smooth functioning of this year's meeting and for coordination with the ICFP 2017 organizers. Karen Barnes helped Sam a great deal with logistic matters. Jamie Vicary, the Workshop Chair, has taken care of the complex logistics involved with selecting and organizing the associated workshops. Jeremy Gibbons, the ICFP 2017 chair, has been very helpful and accommodating in the successful co-location of two conferences. The steering committee, lead by Luke Ong, provided valuable guidance in setting up this meeting and with ensuring that FSCD will have a bright and enduring future. Finally, I thank all participants of the conference for creating a lively and interesting event.

FSCD 2017 benefited from being held "in-cooperation" with the ACM SIGLOG and ACM SIGPLAN and from support from the British Logic Colloquium.

Dale Miller
Program Committee Chair of FSCD 2017

Steering Committee

Thorsten Altenkirch	University of Nottingham	United Kingdom
Sandra Alves (Publicity Chair)	University of Porto	Portugal
Gilles Dowek	Inria	France
Santiago Escobar	University Politecnica de Valencia	Spain
Maribel Fernandez	King's College London	United Kingdom
Hugo Herbelin	Inria	France
Delia Kesner	Université Paris Diderot	France
Naoki Kobayashi	University of Tokyo	Japan
Luke Ong (Chair)	Oxford University	United Kingdom
Brigitte Pientka	McGill University	Canada
René Thiemann	University of Innsbruck	Austria

2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017).
Editor: Dale Miller
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Andreas Abel	Gothenburg University	Sweden
Elvira Albert	Complutense Madrid	Spain
María Alpuente	TU Valencia	Spain
Takahito Aoto	Niigata University	Japan
Zena Ariola	University of Oregon	USA
Federico Aschieri	TU Wien	Austria
Stefano Berardi	University of Turin	Italy
Lars Birkedal	Aarhus University	Denmark
Filippo Bonchi	CNRS & ENS Lyon	France
Pierre Clairambault	CNRS & ENS Lyon	France
Ugo Dal Lago	University of Bologna	Italy
Herman Geuvers	Radboud University	Netherlands
Silvia Ghilezan	University of Novi Sad	Serbia
Jürgen Giesl	RWTH Aachen University	Germany
Hugo Herbelin	Inria	France
Jan Hoffmann	Carnegie Mellon	USA
Deepak Kapur	University of New Mexico	USA
Paul Blain Levy	University of Birmingham	United Kingdom
Dale Miller (chair)	Inria & LIX	France
Paulo Oliva	Queen Mary University of London	United Kingdom
Vincent van Oostrom	University of Innsbruck	Austria
Daniela Petrisan	Université Paris Diderot	France
Femke van Raamsdonk	Vrije Universiteit Amsterdam	Netherlands
Grigore Rosu	University of Illinois	USA
Albert Rubio	UPC-BarcelonaTech	Spain
Paula Severi	University of Leicester	United Kingdom
Bas Spitters	Aarhus University	Denmark
Aaron Stump	The University of Iowa	USA
Kazushige Terui	Kyoto University	Japan
René Thiemann	University of Innsbruck	Austria
Sophie Tison	Lille University	France

2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017).
Editor: Dale Miller
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Beniamino Accattoli
Benedikt Ahrens
Thorsten Altenkirch
Sandra Alves
Puri Arenas
Rob Arthan
David Baelde
Patrick Bahr
Henning Basold
Alessandro Berarducci
Dariusz Biernacki
Valentin Blot
Miquel Bofill
Auke Booij
Cristina Borralleras
Marc Brockschmidt
Ulrik Buchholtz
Guillaume Burel
Simon Castellan
Xiaohong Chen
Jasmin Christian Blanchette
Ranald Clouston
Youyou Cong
Ankush Das
Pietro Di Gianantonio
Paul Downen
Rasmus Ejlers Møgelberg
Martin Escardo
Santiago Escobar
Gilda Ferreira
Francisco Ferreira
Marcelo Fiore
Denis Firsov
Brendan Fong
Jonas Frey
Florian Frohn
Daniil Frumin
Charles Grellois
Giulio Guerrieri
Ferruccio Guidi
Nick Gurski
Raúl Gutiérrez
Makoto Hamana
Jules Hedges
Nao Hirokawa
Naohiko Hoshino
Philip Johnson-Freyd
Sebastiaan Joosten
Stefan Kahrs
Ohad Kammar
Kentaro Kikuchi
Ekaterina Komendantskaya
James Laird
Tomer Libal
Ugo de'Liguoro
Sam Lindley
Salvador Lucas
Víctor López Juan
Ian Mackie
Giulio Manzonetto
Enrique Martin-Martin
Aart Middeldorp
Samuel Mimram
Manuel Montenegro
Naoki Nishida
Adrian Palacios
Dusko Pavlovic
Lucas Pena
Mati Pentus
Jorge A. Pérez
Andrew Pitts
François Pottier
Thomas Powell
Franziska Rapp
Amr Sabry
Masahiko Sakai
Sylvain Salvati
Sam Sanders
Julia Sapiña
Haruhiko Sato
Jakob Grue Simonsen
Silvia Steila
Thomas Sternagel
Christian Sternagel
Sorin Stratulat
Salvador Tamarit
David Thibodeau
Amin Timany
Hugo Torres Vieira
Ayberk Tosun
Yoshihito Toyama
Dmitriy Traytel
Nikos Tzevelekos
Sander Uijlen
Pawel Urzyczyn
Daniele Varacca
Lionel Vaux
Rakesh Verma
Andrea Vezzosi
Pierre Vial
Johannes Waldmann
Freek Wiedijk
Akihisa Yamada
Fabio Zanasi
Hans Zantema
Fer-Jan de Vries

2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017).
Editor: Dale Miller
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Type Systems for the Relational Verification of Higher Order Programs

Marco Gaboardi

University at Buffalo, The State University of New York, Buffalo, NY, USA
gaboardi@buffalo.edu

Abstract

Relational program verification is a variant of program verification where one focuses on guaranteeing properties about the executions of two programs, and as a special case about two executions of a single program on different inputs. Relational verification becomes particularly interesting when non-functional aspects of a computation, like probabilities or resource cost, are considered. Several approached to relational program verification have been developed, from relational program logics to relational abstract interpretation. In this talk, I will introduce two approaches to relational program verification for higher-order computations based on the use of type systems. The first approach consists in developing powerful type system where a rich language of assertions can be used to express complex relations between two programs. The second approach consists in developing more restrictive type systems enriched with effects expressing in a lightweight way relations between different runs of the same program. I will discuss the pros and cons of these two approaches on a concrete example: relational cost analysis, which aims at giving a bound on the difference in cost of running two programs, and as a special case the difference in cost of two executions of a single program on different inputs.

1998 ACM Subject Classification F.3.1 [Logics and Meanings of Programs] Specifying and Verifying and Reasoning about Programs, F.3.2 [Logics and Meanings of Programs] Semantics of Programming Languages

Keywords and phrases Relational verification, refinement types, type and effect systems, complexity analysis

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.1

Category Invited Talk

2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017).
Editor: Dale Miller; Article No. 1; pp. 1:1–1:1

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

Uniform Resource Analysis by Rewriting: Strenghts and Weaknesses

Georg Moser

Department of Computer Science, University of Innsbruck, Innsbruck, Austria
georg.moser@uibk.ac.at

Abstract

In this talk, I'll describe how rewriting techniques can be successfully employed to built state-of-the-art automated resource analysis tools which favourably compare to other approaches. Furthermore I'll sketch the genesis of a uniform framework for resource analysis, emphasising success stories, without hiding intricate weaknesses. The talk ends with the discussion of open problems.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases resource analysis, term rewriting, automation

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.2

Category Invited Talk

1 Prelude

In rewriting, complexity analysis traditionally played a minor role. Complexity analysis was mainly understood as *derivational complexity analysis*, ie. the analysis of the maximal derivation height of a given *first-order term rewrite system* (*TRS* for short), cf. [8, 51]. While deep results, emphasising the connection to proof theory, have been obtained in this direction by Hofbauer, Lautemann and Weiermann, these results were always conceived from the viewpoint of termination analysis, cf. [28, 27, 55]. I exemplarily mention Hofbauer's result that the multiset path order [27] implies primitive recursive derivational complexity, that is, from the proof of termination of TRS $\mathcal{R}$ we can gather the additional information that the length of any derivation over $\mathcal{R}$ starting in a term t will be bounded by a primitive recursive function in the size of t. While in principle a result in runtime complexity analysis of TRSs, this result was proposed by Hofbauer as a classification result of termination analysis: the realm of multiset path orders is restricted to those TRSs of primitive recursive derivation length. See also [54, 16, 18, 53, 34, 36, 40, 35, 39] for further results and pointers into similar research.

Implicit computational complexity theory (ICC for short) provides machine independent characterisations of complexity classes, cf. [19]. While the mainstream in ICC was (and is) concerned with the delineation of suitable restrictions of linear logic, like *bounded linear logic* [24, 20] to characterise the class of polytime computable functions, Cichon and others, followed an approach more in line with related work in rewriting. Suppose program P is abstract and represented as a TRS. Suppose further termination of P follows by a suitable restricted ranking function, more precisely a *restricted polynomial interpretation*. Then in dependence on the precise restrictions enforced the *(worst-case) runtime complexity* of P will be bounded by a polynomial in the size of the input to P. See for example [17, 11, 12, 13].

In conjunction, these traditions provide a wealth of techniques for the runtime complexity analysis of first-order rewrite systems. Due to the focus on automation of termination analysis

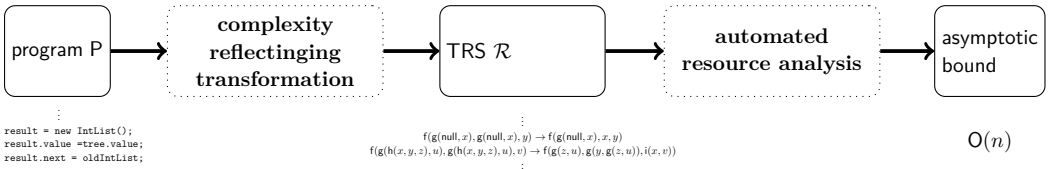


Figure 1 Automated Resource Analysis via Transformations.

in rewriting it was easy to come up with fully automated methods for the derivational and runtime complexity analysis of first-order term rewrite systems, cf. [37, 5].

In this talk, rather than focusing on these techniques precisely, I want to focus on the bigger picture. That is, I will be concerned with application of these methods in the *resource analysis* of programs. Here I aim at a uniform analysis that allows to abstract away particular features of the given program P. In particular we will see ongoing success stories of such a general purpose analysis that can equally well handle *higher-order functional programs* as well as object-oriented bytecode like *Java Bytecode* (*JBC* for short). However, I also want to emphasise challenges and limitations of this approach in practise and discuss future work.

2 Transformation Based Runtime Complexity Analysis

Suppose P is an arbitrary given input program. The idea of resource analysis via *complexity reflecting* transformations is depicted in Figure 1. Here we show the process of transforming a JBC program P into a TRS $\mathcal{R}$, whose runtime complexity is subsequently analysed. Suppose the analysis yields a linear upper bound of the worst-case runtime complexity. Then, as the transformation is *complexity reflecting*, we reflect the analysis of the TRS to conclude $\mathsf{O}(n)$ as asymptotic bound on the runtime complexity of P. If the transformation is also *complexity preserving*, we can employ the same transformation based approach to analyse lower bounds, either for worst-case or best-case complexity. This approach has been successfully evaluated for *higher-order* pure `OCaml` programs, as well as for JBC programs, cf. [4, 38].

With respect to higher-order functional programs the transformation phase also provides a defunctionalisation of higher-order functions in order to represent the original program as an (applicative) term rewrite system. An example run is depicted in Figure 2. The induced whole program analysis, as implemented in our tool T$_\mathsf{C}$T [6], is comparable in strength to the RaML prototype [29, 30] and occasionally stronger. In particular our analysis is able to handle variable capture in closures properly. A noticeable difference between our analysis and RaML are the asymptotic bounds obtained by T$_\mathsf{C}$T, in comparison to the precise bounds by the RaML prototype. This, however is more a design choice, rather than a restriction. Asymptotic bounds allow for better and instant readability and usability, which is at the core of our concern: provide the working programmer with a tool that quickly and automatically provides resource bounds, which can be directly employed for the validation of design choices. Furthermore, it simplifies composability of the analysis.

With respect to imperative programs, our approach allows a more refined representation of the heap in contrast to tools like COSTA [1], CoFloCo [22], or AProVE [23]. This yields tight asymptotic bounds in some cases, which are outside of the realm of the competing methods. These often employ a path-length abstraction of the heap, which is practically more sensible in a variety of cases, but which falls short in programs that crucially rely on an analysis of the heap.

Still, the comparison of the transformational approach to methods rooted more classically in program analysis clarified a shortcoming of the approach. In Figure 1 a unique abstract

(a) Reversing a list, taken from Bird's textbook on functional programming [10].

```
let rec fold_left f acc = function
    [] → acc
  | x::xs → fold_left f (f acc x) xs ;;
let rev l = fold_left (fun xs x → x::xs) [] l ;;
```

(b) Defunctionalised applicative rewrite system.

$$
\begin{aligned}
\mathsf{main}(x_0) &\to \mathsf{m}_1(x_0) \mathbin{@} \mathsf{f} & \mathsf{r}(x_0) \mathbin{@} x_1 &\to x_0 \mathbin{@} \mathsf{r}_1 \mathbin{@} [\,] \mathbin{@} x_1 \\
\mathsf{m}_1(x_0) \mathbin{@} x_1 &\to \mathsf{m}_2(x_0) \mathbin{@} \mathsf{r}(x_1) & \mathsf{r}_1 \mathbin{@} x_0 &\to \mathsf{r}_2(x_0) \\
\mathsf{m}_2(x_0) \mathbin{@} x_1 &\to x_1 \mathbin{@} x_0 & \mathsf{r}_2(x_0) \mathbin{@} x_1 &\to x_1 :: x_0 \\
\mathsf{f} \mathbin{@} x_0 &\to \mathsf{f}_1 \mathbin{@} x_0 & \mathsf{f}_3(x_0, x_1) \mathbin{@} x_2 &\to \mathsf{f}_4(x_2, x_0, x_1) \\
\mathsf{f}_1 \mathbin{@} x_1 &\to \mathsf{f}_2(x_1) & \mathsf{f}_4([\,], x_0, x_1) &\to x_1 \\
\mathsf{f}_2(x_1) \mathbin{@} x_2 &\to \mathsf{f}_3(x_1, x_2) & \mathsf{f}_4(x_0 :: x_1, x_2, x_3) &\to \mathsf{f} \mathbin{@} x_1 \mathbin{@} (x_2 \mathbin{@} x_3 \mathbin{@} x_0) \mathbin{@} x_2
\end{aligned}
$$

(c) Simplified first-order term rewrite system.

$$
\mathsf{main}(x_0) \to \mathsf{f}([\,], x_0) \qquad \mathsf{f}(x_0, [\,]) \to x_0 \qquad \mathsf{f}(x_0, x_1 :: x_2) \to \mathsf{f}(x_1 :: x_0, x_2)
$$

Figure 2 Example run of the `HoCA` prototype on a pure `OCaml` program.

```
public static void test(int n, int m){
  if (0 < n && n < m) {
    int j = n+1;
    while(j < n || j > n){
      if (j > m){
        j=0;
      } else {
        j=j+1;
      }
    }}}
```

Figure 3 The Need for Disjunctive Bounds.

representation was conceived. Either TRSs or a straightforward extension thereof. However, in practise this turned out to be a weakness of the setup, which hinders a competitive resource analysis.

3 Uniform Resource Analysis by Rewriting

To clarify the weakness of assuming one specific form of intermediate language, we consider the Java program depicted in Figure 3. The examples is due to Gulwani et al. [25]. It is not difficult to see that the program is terminating. Given non-zero numbers n, m as input, such that $n < m$, the while loop is executed as long as $j \neq n$ holds, that is, exactly m times. While simple, the example is not trivial, for example, state-of-the-art tools like AProVE or COSTA cannot even show termination fully automatically. Furthermore the straightforward transformation of this program into a TRS, following the translation proposed in [21] is non-terminating.

In order to overcome this, a different flavour of rewrite systems need to be employed as abstract representations. More precisely, the crucial features of an imperative program can be better characterised by variants of transition systems [41, 45]. In particular, Figure 5 depicts

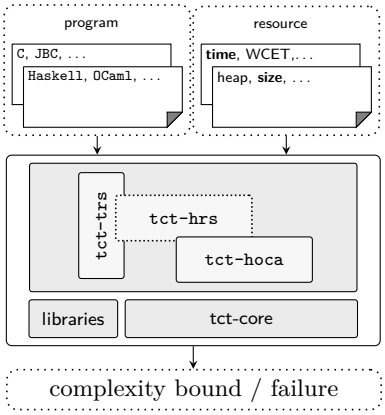


Figure 4 Complexity Analyser T$_C$T.

a suitable encoding of the example in Figure 3 into a so-called *integer transition system* (*ITS* for short). The transformation is complexity reflecting and the runtime complexity can thus be automatically assessed. Our implementation of the resource analysis of ITSs takes inspiration from [15]. However, while transition systems work well for imperative programs, the implicit size abstraction does not provide a suitable match for (higher-order) functional programs, like the list reversal algorithm shown in Figure 2.

We have cast these observations into a uniform resource analysis tool which is largely independent on the specifics of the programming language (and even programming paradigm) of the given input program P. The setup of our tool T$_C$T is depicted in Figure 4. First, the input program in relation to the resource of interest is transformed to an abstract representation, generalising the transformational approach briefly described in Section 2. We refer to the result of applying such a transformation as *abstract program*. It has to be guaranteed that the employed transformations are *complexity reflecting* and if our interest are lower bound also *complexity preserving*. Note that the resource analysis deals with a general *resource analysis problem* that consists of a program together with the resource metric of interest as input. Second, we employ problem specific techniques to derive bounds on the given problem and finally, the result of the analysis, that is, a successful asymptotic resource analysis or the notice of failure, is relayed to the original program. T$_C$T is open source, released under the BSD3 license. All components of T$_C$T are written in *Haskell*. T$_C$T is open with respect to the complexity problem under investigation and problem specific techniques. Moreover it provides an expressive problem independent strategy language that facilitates the proof search, extensibility and automation. The code of T$_C$T, at least for the analysis of term rewrite systems is in part certified, that is, the analysis is guaranteed to provide a correct upper bound, see for example. [7]. A more detailed account of the implementation of T$_C$T and the underlying abstract resource analysis framework giving rise to an uniform resource analysis indicated above can be found in [6].

I want to emphasise that T$_C$T does not make use of a unique abstract representation, but is designed to employ a variety of different representations. Moreover, different representations may interact with each other. For now we make use of ITSs and various forms of rewrite systems, not necessarily first-order TRSs. Currently, we are in the process of developing dedicated techniques for the analysis of *higher-order rewrite systems* (*HRSs* for short) that once should become another abstraction subject to resource analysis (depicted as `tct-hrs` in the figure). To exemplify this, the actual strategy code to analyse object-oriented bytecode

```
start(m,n,j) -> while(m,n,n+1) :|: m > n && n > 0
while(m,n,j) -> while(m,n,0)   :|: m > n && n > 0 && j > n && j > m
while(m,n,j) -> while(m,n,j+1) :|: m > n && n > 0 && j > n && j <= m
while(m,n,j) -> while(m,n,j+1) :|: m > n && n > 0 && j < n && j <= m
```

Figure 5 Integer Transition System.

```
jbc :: Strategy ITS () → Strategy TRS () → Strategy JBC ()
jbc its trs = toCTRS ⋙ Race (toIts ⋙ its) (toTrs ⋙ trs)
```

Figure 6 `Java` Transformation Pipeline Modelled in `tct-jbc`.

programs is shown in Figure 6. Our transformation from `JBC` employs a term abstraction of the heap which gives rise to so-called *constraint term rewrite systems* (*cTRS*). However, in the actual implementation of a resource analysis for cTRS are handled by either conceiving this abstract representation as TRS or as ITS. Consequently we run the correponding method in parallel, as indicated by the keyword `Race` in the code.

The interaction in the resource analysis of these different abstract programs has not yet reached its full potential. But is envisioned in the future to work also on the level of program slices. Certain parts of a program P dealing for example with user-defined datatypes are best abstracted as terms. Other parts are more naturally abstracted in size, like an increasing counter. Based on these different abstractions, different forms of resource analysis are most effective. Combining the obtained different results effectively and in a precise manner is one of our current research agendas. This setup improves *modularity* of the approach and provides scalability and precision of the overall analysis.

Our current work mainly aims at improvements of the backend of T$_C$T, that is, the improvement of the existing resource analysis for the provided abstract representation as well as the incorporation of effective analysis of a particular new representation, the above mentioned HRSs. With respect to TRSs we have recently finalised the incorporation of an amortised analysis for worst-case bounds on the runtime complexity (see also Section 4 below). Furthermore we are in the process of incorporating an amortised analysis providing lower bounds on the best-case complexity of TRSs. Moreover, we are actively developing a new backend for the resource analysis of ITSs. Here the emphasis is on scalability of the methods.

Still, all is not well in the realm of uniform resource analysis tools. Apart from the future work on a better integration of various forms of abstract representations, we noticed in our quest for the incorporation of amortised analysis that we cannot easily mimic simple methods in this generic setting. In particular we have implemented a simple heuristics of the RaML prototype [30] to yield a univariate amortised analysis. While it was not difficult to extend the sophisticated work of Hoffmann et al. from functional programs to TRSs (see [32, 33]) it is less straightforward to obtain the same analysis strength in practise in our uniform setting. I detail the observations from this case study in the next section.

4 Case Study: Amortised Resource Analysis

For clarity, I briefly sketch the potential method of amortised analysis as proposed by Sleator and Tarjan in [48, 50]. The motivation for this analysis technique were self-balancing data structures that sometimes need to perform costly operations that pay off later on, e.g. rebalancing operations on a search tree.

In brief, one assigns to the participating datastructures a nonnegative real-value, the *potential* in an a priori arbitrary fashion. One then defines the amortised cost of an operation

as its actual cost, for example the runtime, plus the difference in potential of all datastructures before and after the operations. In this way, the amortised cost of a costly operation may be small if it results in a big decrease of potential. On the other hand some cheap operations that increase the potential will be overcharged. In this way, one can "save money" now to pay for costly operations later. By a simple telescoping argument the sum of all amortised costs in a sequence of operations plus the potential of the initial input data structure is also an upper bound on the *actual* cost of that sequence. Essentially by reversing the orders, the method can be used equally well for lower bound analysis of best-case complexities.

Thus amortised analysis may yield rigorous bounds on actual resource usage and not just approximate or average bounds. If the potential functions are chosen well then the amortised costs of operations are either constant or exhibit merely a very simple dependency on the maximum size of all intermediate results. A simple classical example is the implementation of a queue by two stacks, an in-tray and an out-tray. Incoming elements are added to the in-tray, outgoing elements are taken from the top of the out-tray. Only if the out-tray becomes empty the entire in-tray is reverse-copied into the out-tray. In this case, the length of the in-tray is clearly a suitable potential function. The costly operation of copying can entirely be paid from the big decrease in potential it causes.

For automated resource analysis, amortised cost analysis has been in particular pioneered by Hoffmann et al., whose RaML prototype has grown into a highly sophisticated analysis tool for (higher-order) functional programs. In a similar spirit, resource analysis tools for imperative programs like COSTA [2], CoFloCo [22] and LOOPUS [47] have integrated amortised reasoning. See also work by Atkey [3].

Let me sketch the implementation of the univariate amortised analysis as proposed in [32]. Naturally the choice of the correct potential functions is usually non-trivial. Following an approach proposed already in [31] we select for each datatype a suitable annotation to define the potential functions. Say these annotations are vectors of natural numbers. For a value v its potential $\Phi(v)$ can then be estimated as a polynomial in the size of v, where the degree of the polynomial depends on the length of the resource annotations used. In automation the precise annotations are unknown and a symbolic amortised cost analysis is employed to fix these annotations. For this one makes use of a simple heuristics [31] taking into account the type signatures of the program considered. The heuristics guarantees that the constraints remain linear, which allows a very efficient analysis, cf. [30].

In the setup of T$_\mathsf{C}$T such an analysis is performed on the abstract representation of the original (first-order) RaML programs. A natural and direct abstraction are TRSs. TRSs are blind on types and we loose in this transformation the seemingly neglectable type information in the input RaML program. Indeed the analysis presented in [32] does not depend on the type information and in [33] we naturally emphasised this generalisation as a step forward. However, without this type information the heuristics doesn't perform well, see Figure 7.

The table presents the results of the amortised resource analysis compared to the analysis using the heuristics discussed above on an independent testbed comprised of a collection of direct encodings of functional programs as well as automatic translations thereof. Ara denotes our standard implementation of amortised analysis in T$_\mathsf{C}$T, which employs non-linear constraints encoded as SMT problems so that either MiniSMT (M) or Z3 (Z) can be employed. On the other hand H indicates the use of heuristics, where the TRSs were manually typed to have comprehensive type information required. As we see the heuristics is significantly faster, but shows significantly poorer behaviour in power. Note that this heuristics works perfectly well within the RaML prototype.

Arguably this is a bug or feature of T$_\mathsf{C}$T. One can correctly argue that we should not strip information from the input language, if we observe that this information is crucial for

	Number of TRSs				Execution Time (in seconds)			
Result	Ara M	Ara Z	Ara MH	Ara ZH	Ara M	Ara Z	Ara MH	Ara ZH
MAYBE	15	14	30	31	13.39	10.27	4.65	6.37
$O(n^1)$	17	17	12	12	0.29	0.31	0.23	0.20
$O(n^2)$	9	9	4	4	5.89	6.40	0.55	0.51
$O(n^3)$	1	1	1	1	1.37	1.58	0.63	0.53
Timeout	16	17	11	10	60.01	60.05	60.02	60.06
Sum/Avg	27	27	17	17	21.04	21.19	13.88	13.85

Figure 7 Experimental Evidence.

```
while (B_s x > b_s) ∧ (B_w x ⩾ b_w)
{
  x' = Ax + c
}
```

Figure 8 Simple Loop Program.

the effectivity of the resource analysis. However, by doing so we effectively instantiate RaML programs as yet another abstract representation thereby completely loosing the uniformity of our tool. As typed TRS are not standard we cannot employ the method for the analysis of TRS proper. In my understanding this is not a sensible approach but it clarifies that a uniform solver like TcT may easily become too powerful to be of any use, if not engineered well.

Similar cases can be made for imperative programs and the comparison to cost equations as abstract representations. I believe that such seemingly limitations of the here proposed methodology may act as stepping stone to a more intense coupling of different techniques in one solver so that to achieve the best of all worlds in resource analysis.

5 Open Problems

In the following I want to mention two open problems directly related to the effective resource analysis of programs, whose solutions appears to require some thought.

Our ongoing quest for automated resource analysis of term rewrite systems are essentially fuelled by the idea to automatically perform amortised resource analysis as proposed in standard textbooks [43]. All automated methods provided so far in the literature are restricted to functions with *constant* amortised costs [30], while a convincing argument can be made for the automated analysis of algorithms with *logarithmic* amortised costs. In particular a number of classical examples of amortised analysis for self-balancing datastructures have logarithmic amortised costs. It is currently very unclear how to tackle this problem, raised as a challenge by Nipkow last year, cf. [42].

Yet another area of related study are *simple loop programs*. Simple loop programs are typically represented as matrix programs of the following form depicted in Figure 8. Here $B_s\mathbf{x} > \mathbf{b_s}$ and $B_w\mathbf{x} \geqslant \mathbf{b_w}$ represent conjunctions of linear strict or weak inequalities over the state variables $\mathbf{x}$. Furthermore $A\mathbf{x} + \mathbf{c}$ represents an affine update of each state variable. Execution of the instruction in the loop body is in parallel, which is denoted as usual by the use of primed state variables. Simple loop programs have also be called *linear while loop programs* in the literature. If vectors $\mathbf{b_s}, \mathbf{b_w}$ and $\mathbf{c}$ are zero vectors then the program is called *homogeneous*, otherwise it is *inhomogeneous*.

State variables are either interpreted over the reals, the rationals, or over the integer and depending on the domain or further restrictions on the programs, termination is known to be decidable. More precisely over $\mathbb{R}$ and $\mathbb{Q}$ termination is decidable in polynomial time [52, 14]. For homogenous programs over $\mathbb{N}$ these results imply decidability. In the case that the update matrix A is diagonalisable, that is, if it is similar to a diagonal matrix, then termination for integer loop programs is decidable, cf. [44]. However the general case for integer loop programs is (wide) open and in practise are typically subject to ranking functions [46, 9].

Building upon these decidability results we focus on a resource analysis of simple loop programs. In particular we study runtime and size complexity of simple loop programs over $\mathbb{Q}$. Here runtime complexity measures the maximal number of symbolic executions, that is, the number of loops in relation to the value of the input. To date no a priori runtime complexity analysis of simple loop programs is known, although for homogenous programs over $\mathbb{N}$ we have decidability of termination. Due to the strong links of the problem to linear recurrence sequences in general and the Skolem Problem [26, 49] no simple solutions seem forthcoming.

References

1 E. Albert, P. Arenas, S. Genaim, G. Puebla, and D. Zanardini. Cost analysis of object-oriented bytecode programs. *TCS*, 413(1):142–159, 2012.

2 E. Albert, S. Genaim, and A. N. Masud. On the inference of resource usage upper and lower bounds. *TOCL*, 14(3):22, 2013.

3 R. Atkey. Amortised Resource Analysis with Separation Logic. *LMCS*, 7(2), 2011.

4 M. Avanzini, U. Dal Lago, and G. Moser. Analysing the complexity of functional programs: Higher-order meets first-order. In *Proc. 20th ICFP*, pages 152–164. ACM, 2015.

5 M. Avanzini and G. Moser. Tyrolean Complexity Tool: Features and Usage. In *Proc. 24th RTA*, volume 21 of *LIPIcs*, pages 71–80, 2013.

6 M. Avanzini, G. Moser, and M. Schaper. TcT: Tyrolean Complexity Tool. In *Proc. 22nd TACAS*, LNCS, pages 407–423, 2016.

7 M. Avanzini, C. Sternagel, and R. Thiemann. Certification of complexity proofs using ceta. In *Proc. 26th RTA*, volume 36 of *LIPIcs*, pages 23–39, 2015.

8 F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

9 A. Ben-Amram and S. Genaim. Ranking functions for linear-constraint loops. *JACM*, 61(4):26:1–26:55, 2014. doi:10.1145/2629488.

10 R. Bird. *Introduction to Functional Programming using Haskell, Second Edition*. Prentice Hall, 1998.

11 G. Bonfante, A. Cichon, J.-Y. Marion, and H. Touzet. Complexity classes and rewrite systems with polynomial interpretations. In *Proc. 7th CSL*, volume 1584 of *LNCS*, pages 372–384, 1998.

12 G. Bonfante, A. Cichon, J.-Y. Marion, and H. Touzet. Algorithms with polynomial interpretation termination proof. *JFP*, 11(1):33–53, 2001.

13 G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. Quasi-interpretations a way to control resources. *TCS*, 412(25):2776–2796, 2011.

14 M. Braverman. Termination of integer linear programs. In *Proc. 18th CAV*, volume 4144 of *LNCS*, pages 372–385, 2006.

15 M. Brockschmidt, F. Emmes, S. Falke, C. Fuhs, and J. Giesl. Alternating runtime and size complexity analysis of integer programs. In *Proc. 20th TACAS*, volume 8413 of *LNCS*, pages 140–155, 2014.

16 W. Buchholz. Proof-theoretical analysis of termination proofs. *APAL*, 75:57–65, 1995.

17 E.-A. Cichon and P. Lescanne. Polynomial interpretations and the complexity of algorithms. In *Proc. 11th CADE*, volume 607 of *LNCS*, pages 139–147, 1992.

18 E.-A. Cichon and A. Weiermann. Term rewriting theory for the primitive recursive functions. *APAL*, 83(3):199–223, 1997.

19 U. Dal Lago. A short introduction to implicit computational complexity. In *Lectures on Logic and Computation – ESSLLI 2010*, volume 7388 of *LNCS*, pages 89–109, 2011.

20 U. Dal Lago and M. Hofmann. Bounded linear logic, revisited. *LMCS*, 6(4), 2010.

21 S. Falke and D. Kapur. A term rewriting approach to the automated termination analysis of imperative programs. In *Proc. 22nd CADE*, volume 5663 of *LNCS*, pages 277–293, 2009.

22 A. Flores-Montoya. Upper and lower amortized cost bounds of programs expressed as cost relations. In *Proc. 21st FM*, volume 9995 of *LNCS*, 2016. doi:10.1007/978-3-319-48989-6_16.

23 J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. Analyzing program termination and complexity automatically with aprove. *JAR*, 58(1):3–31, 2017. doi:10.1007/s10817-016-9388-y.

24 J.-Y. Girard, A. Scedrov, and P. Scott. Bounded linear logic: A modular approach to polynomial-time computability. *TCS*, 97(1):1–66, 1992.

25 S. Gulwani and F. Zuleger. The reachability-bound problem. In *Proc. PLDI'10*, pages 292–304. ACM, 2010.

26 V. Halava, T. Harju, M. Hirvensalo, and J. Karhumäki. *Skolem's Problem: On the Border Between Decidability and Undecidability*. TUCS technical report. Turku Centre for Computer Science, 2005.

27 D. Hofbauer. Termination proofs by multiset path orderings imply primitive recursive derivation lengths. *TCS*, 105:129–140, 1992.

28 D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In *Proc. 3rd RTA*, volume 355 of *LNCS*, pages 167–177, 1989.

29 J. Hoffmann, K. Aehlig, and M. Hofmann. Multivariate amortized resource analysis. *TOPLAS*, 34(3):14, 2012.

30 J. Hoffmann, A. Das, and S. Weng. Towards automatic resource bound analysis for OCaml. In *Proc. 44th POPL*, pages 359–373. ACM, 2017.

31 J. Hoffmann and M. Hofmann. Amortized resource analysis with polynomial potential. In *Proc. 19th ESOP*, volume 6012 of *LNCS*, pages 287–306, 2010.

32 M. Hofmann and G. Moser. Amortised resource analysis and typed polynomial interpretations. In *Proc. of Joint 25th RTA and 12th TLCA*, volume 8560 of *LNCS*, pages 272–286, 2014.

33 M. Hofmann and G. Moser. Multivariate amortised resource analysis for term rewrite systems. In *Proc. 13th TLCA*, volume 38 of *LIPIcs*, pages 241–256, 2015. doi:10.4230/LIPIcs.TLCA.2015.241.

34 I. Lepper. Derivation lengths and order types of Knuth-Bendix orders. *TCS*, 269:433–450, 2001.

35 A. Middeldorp, G. Moser, F. Neurauter, J. Waldmann, and H. Zankl. Joint spectral radius theory for automated complexity analysis of rewrite systems. In *Proc. 4th CAI*, volume 6742 of *LNCS*, pages 1–20, 2011.

36 G. Moser. The Hydra battle and Cichon's principle. *AAECC*, 20(2):133–158, 2009. doi:10.1007/s00200-009-0094-4.

37 G. Moser. Proof theory at work: Complexity analysis of term rewrite systems. *CoRR*, abs/0907.5527, 2009. Habilitation Thesis.

38 G. Moser and M. Schaper. A complexity preserving transformation from Jinja bytecode to rewrite systems. *IC*, 2017. To appear.

39 G. Moser and A. Schnabl. The derivational complexity induced by the dependency pair method. *LMCS*, 7(3), 2011.

40 G. Moser and A. Schnabl. Termination proofs in the dependency pair framework may induce multiple recursive derivational complexity. In *Proc. 21st RTA*, volume 10 of *LIPIcs*, pages 235–250, 2011.

41 F. Nielson, H. R. Nielson, and C. Hankin. *Principles of program analysis*. Springer, 2005.

42 T. Nipkow. Verified analysis of functional data structures. In *1st FSCD*, pages 4:1–4:2, 2016. doi:10.4230/LIPIcs.FSCD.2016.4.

43 C. Okasaki. *Purely functional data structures*. Cambridge University Press, 1999.

44 J. Ouaknine, J. S. Pinto, and J. Worrell. On termination of integer linear loops. In *Proc. 26th SODA*, pages 957–969. SIAM, 2015.

45 A. Podelski and A. Rybalchenko. Transition predicate abstraction and fair termination. *TOPLAS*, 29(3), 2007.

46 A. Podelski and Andrey Rybalchenko. A complete method for the synthesis of linear ranking functions. In *Proc. 5th VMCAI*, volume 2937 of *LNCS*, pages 239–251, 2004.

47 M. Sinn, F. Zuleger, and H. Veith. A simple and scalable static analysis for bound analysis and amortized complexity analysis. In *Proc. Software Engineering*, volume 252 of *LNI*, pages 101–102, 2016.

48 D. Sleator and R. Tarjan. Self-adjusting binary trees. In *Proc. of the 15th STOC*, pages 235–245. ACM, 1983. doi:10.1145/800061.808752.

49 T. Tao. *Structure and randomness: pages from year one of a mathematical blog*. AMS, 2008.

50 R. E. Tarjan. Amortized computational complexity. *SIAM J. Alg. Disc. Meth*, 6(2):306–318, 1985.

51 TeReSe. *Term Rewriting Systems*, volume 55 of *Cambridge Tracks in Theoretical Computer Science*. Cambridge University Press, 2003.

52 A. Tiwari. Termination of linear programs. In *Proc. 16th CAV*, volume 3114 of *LNCS*, pages 70–82, 2004.

53 H. Touzet. Encoding the Hydra battle as a rewrite system. In *Proc. 23rd MFCS*, volume 1450 of *LNCS*, pages 267–276, 1998.

54 A. Weiermann. Complexity bounds for some finite forms of Kruskal's theorem. *JSC*, 18(5):463–488, November 1994.

55 A. Weiermann. Termination proofs for term rewriting systems with lexicographic path ordering imply multiply recursive derivation lengths. *TCS*, 139:355–362, 1995.

Brzozowski Goes Concurrent – A Kleene Theorem for Pomset Languages

Alexandra Silva

University College London, London, UK
alexandra.silva@ucl.ac.uk

Abstract

Concurrent Kleene Algebra (CKA) is a mathematical formalism to study programs that exhibit concurrent behaviour. As with previous extensions of Kleene Algebra, characterizing the free model is crucial in order to develop the foundations of the theory and potential applications. For CKA, this has been an open question for a few years and this talk will overview why the problem is so difficult. We will then pave the way towards a solution, by presenting a new automaton model and a Kleene-like theorem for CKA. More precisely, we connect a relaxed version of CKA to series-parallel pomset languages, which are a natural candidate for the free model. There are two substantial differences with previous work: from expressions to automata, we use Brzozowski derivatives, which enable a direct construction of the automaton; from automata to expressions, we provide a syntactic characterization of the automata that denote valid CKA behaviours. We also survey how the present work can be used to to extend the network specification language NetKAT with primitives for concurrency so as to model and reason about concurrency within networks. This is joint work with Tobias Kappe, Paul Brunet, Bas Luttik, and Fabio Zanasi.

1998 ACM Subject Classification F.3.1 [Logics and Meanings of Programs] Specifying and Verifying and Reasoning about Programs, F.3.2 [Logics and Meanings of Programs] Semantics of Programming Languages

Keywords and phrases Kleene algebras, Pomset languages, concurrency, NetKAT

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.3

Category Invited Talk

Quantitative Semantics for Probabilistic Programming

Christine Tasson

IRIF, Université Paris Diderot – Paris 7, Paris, France
tasson@irif.fr

Abstract

Probabilistic programming has many applications in statistics, physics,... so that all programming languages have been equipped with probabilistic library. However, there is a need in developing semantical tools in order to formalize higher order and recursive probabilistic languages. Indeed, it is well known that categories of measurable spaces are not Cartesian closed. We have been studying quantitative semantics of probabilistic spaces to fill this gap. A first step has been to focus on probabilistic programming languages with discrete types such as integers and booleans. In this setting, probabilistic programs can be seen as linear combinations of deterministic programs. Probabilistic Coherent Spaces constitute a Cartesian closed category that is fully abstract with respect to probabilistic Call-By-Push-Value. Moreover, this toy language is endowed with a memorization operator that allow to encode most discrete probabilistic programs. The second step is to move on probabilistic programming with continuous types representing for instance reals endowed with Lebesgue measurable sets. We introduce the category of cones and stable functions which is Cartesian closed. The trick is to enlarge the category of measurable spaces to gain closeness and to embrace measurable spaces. Besides, the category of cones is a sound and adequate model of a higher order and recursive probabilistic language in which most classical distributions and probabilistic tools can be encoded. This is joint work with Thomas Ehrhard and Michele Pagani.

1998 ACM Subject Classification F.3.2 Semantics of Programming Language

Keywords and phrases denotational semantics, probabilistic programming, programming language, probability

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.4

Category Invited Talk

2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017).
Editor: Dale Miller; Article No. 4; pp. 4:1–4:1

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

Displayed Categories*

Benedikt Ahrens[1] and Peter LeFanu Lumsdaine[2]

1 **Inria, Nantes, France**
`benedikt.ahrens@inria.fr`
2 **Department of Mathematics, Stockholm University, Stockholm, Sweden**
`p.l.lumsdaine@math.su.se`

Abstract

We introduce and develop the notion of *displayed categories*.

A displayed category over a category $\mathcal{C}$ is equivalent to 'a category $\mathcal{D}$ and functor $F : \mathcal{D} \to \mathcal{C}$', but instead of having a single collection of 'objects of $\mathcal{D}$' with a map to the objects of $\mathcal{C}$, the objects are given as a family indexed by objects of $\mathcal{C}$, and similarly for the morphisms. This encapsulates a common way of building categories in practice, by starting with an existing category and adding extra data/properties to the objects and morphisms.

The interest of this seemingly trivial reformulation is that various properties of functors are more naturally defined as properties of the corresponding displayed categories. Grothendieck fibrations, for example, when defined as certain functors, use equality on objects in their definition. When defined instead as certain displayed categories, no reference to equality on objects is required. Moreover, almost all examples of fibrations in nature are, in fact, categories whose standard construction can be seen as going via displayed categories.

We therefore propose displayed categories as a basis for the development of fibrations in the type-theoretic setting, and similarly for various other notions whose classical definitions involve equality on objects.

Besides giving a conceptual clarification of such issues, displayed categories also provide a powerful tool in computer formalisation, unifying and abstracting common constructions and proof techniques of category theory, and enabling modular reasoning about categories of multi-component structures. As such, most of the material of this article has been formalised in Coq over the `UniMath` library, with the aim of providing a practical library for use in further developments.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Category theory, Dependent type theory, Computer proof assistants, Coq, Univalent mathematics

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.5

1 Introduction

It is often said that reference to equality of objects of categories is in general both undesirable and unnecessary.

* This material is based upon work supported by the National Science Foundation under agreement No. DMS-1128155 and CMU 1150129-338510. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work has partly been funded by the CoqHoTT ERC Grant 637339, and by the Swedish Research Council (VR) Grant 2015-03835 *Constructive and category-theoretic foundations of mathematics*.

There are some topics, however, whose development does appear to require it. One example often given is the definition of *(Grothendieck) fibrations* (and their relatives): functors $p : \mathcal{D} \to \mathcal{C}$ equipped with a lifting property providing (among other things) an object d of $\mathcal{D}$ such that pd is equal to a previously given object c of $\mathcal{C}$. A similar example is the property of *creating limits*; see [8, Remark 5.3.7] for an explicit discussion of this example.[1]

In examples of fibrations (or creation of limits), however, one virtually never has cause to speak explicitly of equality of objects; and equally in their basic general theory.

How is this avoidance achieved? In the general development, equality occurs only within the notion of 'objects of $\mathcal{D}$ over c', for objects c of $\mathcal{C}$. And in examples, there is almost always an obvious alternative notion of 'object $\mathcal{D}$ over c', trivially equivalent to 'objects of $\mathcal{D}$ whose projection is equal to c', but expressible without mentioning equality of objects.

Specifically, objects of $\mathcal{D}$ typically consist of objects of $\mathcal{C}$ equipped with extra structure or data; 'an object of $\mathcal{D}$ over c' is then understood to mean 'a choice of the extra data for c'. For instance, in showing that the forgetful functor $\mathsf{grp} \to \mathsf{set}$ creates limits, one doesn't construct a group and then note that its underlying set is the desired one; one simply constructs a group structure on the set.

The notion of *displayed categories* makes this explicit. A displayed category over $\mathcal{C}$ consists of a family of types $\mathcal{D}_c$ (of 'objects over c'), indexed by objects c of $\mathcal{C}$, and similarly sets of morphisms indexed by morphisms of $\mathcal{C}$, along with suitable composition and identity operations to ensure that the total collections of objects and morphisms form a category (with a projection functor to $\mathcal{C}$). This is entirely equivalent to the data of a category with a functor to $\mathcal{C}$, just as 'a family of sets indexed by X' is equivalent to 'a set with a function to X'.

If fibrationhood (or creating limits, etc.) is now defined not as a property of a functor but instead as a property of a displayed category, no mention of equality of objects is required. Equality of objects is used only for turning an arbitrary functor into a displayed category; but this is rarely needed in practice, since most natural examples of fibrations already arise from displayed categories. For instance, the standard definition of the category of groups can be read as the total category of the displayed category over set whose objects are group structures on sets.

We therefore propose that displayed categories should be taken as a basis for the development of fibrations (and creation of limits, and various other notions), in particular in the type-theoretic setting, where dealing with equality on objects is more practically problematic than in classical foundations.

We do not believe we are introducing something mathematically novel here; we are simply making explicit an aspect of how mathematicians already deal with certain kinds of examples in practice. The payoffs, however, are twofold.

Firstly, since this concept has been previously un-articulated, it has not been consistently appreciated that it resolves the 'problematic' issue of fibrations (and various other notions) apparently requiring use of equality on objects. Besides providing conceptual clarification, this should help in future work with disentangling which constructions genuinely *do* require use of equality on objects, and may require extra work or assumptions to develop in type-theoretic settings.

Secondly, by making this common informal technique precise, we make it available for use in computer formalisation, where a difference between the formal definitions given and the

[1] Both of these definitions have analogues in which the equality is weakened to isomorphism; but the strict versions have nonetheless remained in more general currency.

approach used in practice cannot be so blithely elided as it can for human mathematicians. To that end, most constructions and results of the present paper have been formalised in the proof assistant Coq, over the `UniMath` library, with the goal of providing a practical library for re-use in further developments.

While that development is in univalent type theory, for the present article we work in an 'agnostic' logical setting: all results may be understood either in type theory with univalence, or in a classical set-theoretic foundation.

1.1 Outline

We begin, in § 2, by laying out precisely the agnostic type-theoretic foundation in which we work, and recalling the basic background of category theory in this setting.

In § 3, we then set up the core definitions and constructions of displayed categories, along with various examples which will be used as running illustrations through the following sections.

Following this, in § 4, we consider *creation of limits*, a first simple example of a classical property of functors which can be stated and developed more cleanly as a property of displayed categories.

In § 5, we move to the central such example: fibrations, along with their cousins isofibrations, discrete fibrations, and so on. We set out the displayed-category definitions of these, and set out some of the basic results and constructions over this definition.

This provides a basis for the theory and application of fibrations in the type-theoretic setting. In § 6, we use this to define *comprehension categories* – a categorical axiomatisation of type dependency – bringing together several of the tools set up in earlier sections.

Finally, in § 7, we consider *univalence* of displayed categories. The main result there is that the total category of a univalent displayed category (suitably defined) over a univalent base category is univalent. This generalises the *structure identity principle* of [9, §9.8].

Throughout the article, many proofs would be almost word-for-word the same as standard proofs of the corresponding results about classically-defined fibrations (resp. creation of limits, etc), since displayed categories are exactly a formal abstraction of the language already used in such proofs. We therefore omit these, to avoid repeating well-known material – but we invite the reader to recall the standard proofs, and see how directly they transfer.

Most other proofs are also either omitted or just briefly sketched, if they are either routine, available in detail in the formalisation, or both.

We follow Voevodsky in writing 'Problem', rather than 'Theorem', 'Proposition', etc., to denote proof-relevant results.

1.2 Formalisation

Most results of the present article have been formalised in Coq, over the `UniMath` library of Voevodsky et al. [10].

The primary goal of the formalisation is to provide a library for use in further work. We have therefore focused in it on the results and constructions we expect to be useful in such work. In particular, we have not formalised the comparisons with classical definitions: these are not needed for the development of fibrations etc. based on displayed categories, but rather form a justification that this approach is 'correct' from a classical point of view.

The full code of the formalisation is available on Github, at `https://github.com/UniMath/TypeTheory`, in the subdirectory `TypeTheory/Displayed_Cats`. Instructions for

installation and compilation can be found in the repository's `README.md` file. A browsable version is available at `https://unimath.github.io/TypeTheory/`.

Definitions, constructions, and results included in the formalisation are labelled below with their corresponding identifiers, as e.g. `disp_cat`. The main branch of the formalisation remains under development, so these identifiers may change in future. However, the version current at time of writing will remain permanently available under the tag `2017-displayed-cats-fscd`.

The material of the present paper constitutes about 5,000 lines of code.

2 Background

2.1 Logical setting

All the material of the present paper may be understood either in the univalent setting, or in classical set-theoretic foundations.

Precisely, our background setting throughout is Martin-Löf's intensional type theory, with: Σ-types, with the strong η rule; identity types; Π-types, also with η, and functional extensionality; $\mathsf{0}$, $\mathsf{1}$, $\mathsf{2}$, and N; propositional truncation; and two universes closed under all these constructions.

This setting is *agnostic* about equality on types: it assumes neither univalence, nor UIP. It is therefore expected to be compatible both with the addition of univalence, and with the interpretation of types as classical sets.

Some type-theoretic issues trivialise under the classical reading – for instance, the consideration of transport along equalities, which is unnecessary classically. Some topics also become less interesting there, as they admit only degenerate examples: in particular, the material on univalent categories. The reader interested only in the classical setting may therefore ignore these aspects.

2.2 Type-theoretic background

We mostly follow the terminology standardised in the HoTT book [9]. A brief, but sufficient, overview is given in [2], among other places.

We depart from it (and type-theoretic tradition in general) in writing just *existence* for what is called *mere existence* in [9], since this is what corresponds (under the interpretation of types as sets) to the standard mathematical usage of *existence*.

We will make frequent use of *dependent paths/equalities* [9, §6.2] Specifically, in a type family B_x indexed by $x : A$, we will write dependent equalities as e.g. $p : y_0 =_e y_1$, where $e : x_0 =_A x_1$ and $y_i : B_{x_i}$. We omit explicit mention of the type family B, since it will always be clear from context. The base A will often moreover be a set, in which case $y_0 =_e y_1$ does not depend on the base path e, so we suppress this and write just $y_0 =_* y_1$.

We will mostly ignore size issues; we would really like to think of everything as being universe-polymorphic. For concreteness, however, type may be understood always as the smaller of our two assumed universes, with types in this universe referred to as *small*, and similarly set as meaning the type or category of small sets, and so on.

2.3 Categories

We mostly follow the approach to category theory in the type-theoretic setting established in [2]. We depart however from their terminology, writing *categories* for what [2] calls precat-

egories (since it is these that becomes the standard definition under the set interpretation), and writing *univalent categories* for what [2] calls categories.

Specifically, in a *category* $\mathcal{C}$, the hom-sets $\mathcal{C}(a,b)$ are required to be sets, but the type $\mathcal{C}_0$ of objects is allowed to be an arbitrary type. A category $\mathcal{C}$ is *univalent* if for all $a, b : \mathcal{C}$, the canonical map $\mathsf{idtoiso}_{a,b} : (a = b) \to \mathsf{Iso}_{\mathcal{C}}(a,b)$ is an equivalence: informally, if 'equality of objects is isomorphism in $\mathcal{C}$'.

Following the `UniMath` library, we write composition in the 'diagrammatic' order; that is, the composite of $f : a \to b$ and $g : b \to c$ is denoted $f \cdot g : a \to c$.

3 Displayed categories

In this section, we set out the basic definitions of displayed categories, displayed functors, and displayed natural transformations, along with key constructions on them, and examples which will act as running illustrations throughout the paper.

3.1 Definition and examples

▶ **Definition 1** (`disp_cat`). Given a category $\mathcal{C}$, a *displayed category* $\mathcal{D}$ *over* $\mathcal{C}$ consists of

- for each object $c : \mathcal{C}$, a type $\mathcal{D}_c$ of 'objects over c';
- for each morphism $f : a \to b$ of $\mathcal{C}$, $x : D_a$ and $y : D_b$, a set of 'morphisms from x to y over f', denoted $\mathsf{hom}_f(x,y)$ or $x \to_f y$;
- for each $c : \mathcal{C}$ and $x : \mathcal{D}_c$, a morphism $1_x : x \to_{1_c} x$;
- for all morphisms $f : a \to b$ and $g : b \to c$ in $\mathcal{C}$ and objects $x : \mathcal{D}_a$ and $y : \mathcal{D}_b$ and $z : \mathcal{D}_c$, a function

$$\mathsf{hom}_f(x,y) \times \mathsf{hom}_g(y,z) \to \mathsf{hom}_{f \cdot g}(x,z)\,,$$

denoted like ordinary composition by $(\bar{f}, \bar{g}) \mapsto \bar{f} \cdot \bar{g} : x \to_{f \cdot g} z$, where $\bar{f} : x \to_f y$ and $\bar{g} : y \to_g z$,

such that, for all suitable inputs, we have:

- $\bar{f} \cdot 1_y =_* \bar{f}$,
- $1_x \cdot \bar{f} =_* \bar{f}$,
- $\bar{f} \cdot (\bar{g} \cdot \bar{h}) =_* (\bar{f} \cdot \bar{g}) \cdot \bar{h}$.

Note that the axioms are all *dependent* equalities, over equalities of morphisms in $\mathcal{C}$: for instance, if $\bar{f} : x \to_f y$, then $\bar{f} \cdot 1_y : x \to_{f \cdot 1_b} y$, so the displayed right unit axiom $\bar{f} \cdot 1_y =_* \bar{f}$ is over the ordinary right unit axiom $f \cdot 1_b = f$ of $\mathcal{C}$. This will be typical in what follows: equations in displayed categories will be modulo analogous equations in $\mathcal{C}$, which we will usually suppress without further comment.

As promised, any displayed category over $\mathcal{C}$ induces an ordinary category over $\mathcal{C}$:

▶ **Definition 2** (`total_category`, `pr1_category`). Let $\mathcal{D}$ be a displayed category $\mathcal{D}$ over $\mathcal{C}$. The *total category* of $\mathcal{D}$, written $\int \mathcal{D}$ (or $\int_{\mathcal{C}} \mathcal{D}$, or $\int_{c:\mathcal{C}} \mathcal{D}_c$) is defined as follows:

- objects are pairs (a,x) where $a : \mathcal{C}$ and $x : \mathcal{D}_a$; in other words, the type of objects is

$$(\textstyle\int \mathcal{D})_0 := \sum_{a:\mathcal{C}} \mathcal{D}_a\,,$$

- morphisms $(a,x) \to (b,y)$ are pairs $(f, \bar{f})$ where $f : a \to b$ and $\bar{f} : x \to_f y$; in other words,

$$(\textstyle\int \mathcal{D})\big((a,x),(b,y)\big) := \sum_{f:\mathcal{C}(a,b)} \mathsf{hom}_f(x,y)\,,$$

- composition and identities in $\int \mathcal{D}$ are induced straightforwardly from those of $\mathcal{C}$ and $\mathcal{D}$, and similarly for the axioms.

The evident *forgetful functor* $\pi_1^{\mathcal{D}} : \int \mathcal{D} \to \mathcal{C}$ simply takes the first projection, on both objects and morphisms.

▶ **Example 3** (`disp_grp`)**.** The *category of groups* can be defined as the total category of a displayed category grp, over set:

- grp_X is the set of group structures on the set X;
- given a function $f : X \to Y$ and group structures (μ, e) on X and (μ', e') on Y, $\mathsf{hom}_f((\mu, e), (\mu', e'))$ is (the type representing) the proposition 'f is a homorphism with respect to (μ, e), (μ', e')';
- the displayed composition 'operation' is the fact that the composite of homomorphisms is a homomorphism; similarly for the identity;
- the axioms are trivial, since the displayed hom-sets are propositions.

The total category of this is exactly the usual category of groups.

▶ **Example 4** (`disp_top`)**.** The category of topological spaces can be defined as the total category of the displayed category top over set:

- top_X is the set of topologies on the set X;
- given a function $f : X \to Y$ and topologies T on X and T' on Y, $\mathsf{hom}_f(T, T')$ is the proposition 'f is continuous with respect to X and Y'.

▶ **Example 5** (`disp_over_unit`)**.** Any category can be viewed as a displayed category over the terminal category.

▶ **Example 6** (`disp_full_sub`)**.** Let $P : \mathcal{C} \to \mathsf{type}$ be a (type-valued) predicate on the objects of $\mathcal{C}$. Then there is an associated displayed category, with object family exactly P, and with $\mathsf{hom}_f(y, y') := 1$ for all $f : x \to x'$, $y : Px$, and $y' : Px'$. The operations and axioms are trivial.

Its total category is the full subcategory of $\mathcal{C}$ on objects satisfying the predicate P.

The preceding few examples are instances of a common situation:

▶ **Proposition 7** (`full_pr1_category`, `faithful_pr1_category`, `fully_faithful_pr1_category`)**.** *Let $\mathcal{D}$ be a displayed category over $\mathcal{C}$. If every displayed hom-set $\mathsf{hom}_f(y, y')$ of $\mathcal{D}$ is a proposition (resp. inhabited, contractible) then $\pi_1 : \int \mathcal{D} \to \mathcal{C}$ is faithful (full, fully faithful).* ◀

Besides the total category, a displayed category also possesses *fibre* categories:

▶ **Definition 8** (`fiber_category`)**.** Given a displayed category $\mathcal{D}$ over $\mathcal{C}$, and an object $c : \mathcal{C}$, define the *fibre category* $\mathcal{D}_c$ of $\mathcal{D}$ over c as the category with objects $\mathcal{D}_c$ and with morphisms $\mathsf{hom}(x, y) := \mathsf{hom}_{1_c}(x, y)$. Composition and identity are induced by that of $\mathcal{D}$.

In general these may not be so well behaved as the total category; they will typically be interesting and well-behaved just when $\mathcal{D}$ is an isofibration (Def. 30).

▶ Remark. In choosing notation and terminology for examples of displayed categories, a question arises: should one name displayed categories according to their total category, or according to their fibres?

This problem arises already with fibrations in the classical setting; so we follow for the most part the usual compromises used there. Specifically, when a given total category has a particularly canonical displaying—for example, groups displayed over sets—we will use the

same name for the displayed category and its total category, so for example $G : \mathsf{grp}$ denotes a group, while $(\mu, e) : \mathsf{grp}_X$ is a group structure on X. On the other hand, when different displayed categories have equivalent total categories—for instance, the product $\mathcal{C} \times \mathcal{C}'$ may be displayed over either $\mathcal{C}$ or $\mathcal{C}'$—then we will adopt different notation to distinguish these, usually based on the resulting fibre categories.

Other examples we will meet below include:

- any product $\mathcal{C} \times \mathcal{C}'$, displayed over its first factor as $\mathsf{const}_{\mathcal{C}}\, \mathcal{C}'$ (Example 14);
- the arrow category $\mathcal{C}^{\to}$, in several ways: displayed over $\mathcal{C}^2$, with fibres hom-sets; and displayed over $\mathcal{C}$, with fibres either the slices or the coslices of $\mathcal{C}$ (Example 16);
- categories of algebras for endofunctors and monads (Examples 17, 18).

We postpone their full definitions until we have a few more tools set up.

▶ Remark. Definitions very closely equivalent to that of displayed categories—that is, "fibred" presentations of a functor into a fixed base category—can also be recovered from more sophisticated categorical structures in several ways: as a lax 2-functor or double functor from the base category into the bicategory or double category of spans, or as a normal lax 2-functor/double functor into the bicategory/double category of distributors (as observed by Bénabou in [4, §7]), or as a double profunctor from the base category to the terminal double category.[2]

3.2 Displayed functors and natural transformations

Another occurrence of equality of objects is in various definitions where diagrams of functors are assumed to commute on the nose. For instance, comprehension categories involve a fibration $p : \mathcal{T} \to \mathcal{C}$, and a functor $\chi : \mathcal{T} \to \mathcal{C}^{\to}$, such that $\chi \cdot \mathsf{cod} = p$ [7, Theorem 9.3.4]; similar conditions occur in the definition of functorial factorisations, in the theory of weak factorisation systems (among many other places). It is typically clear that the definitions could also be phrased without equality of objects, at some cost in concision and clarity.

However, they are almost always of the form $G \cdot \pi_1^{\mathcal{D}} = F$, where G is a functor into the total category of some displayed category, and F is a previously-given functor into the base. Indeed, they are often of the more specialised form $G \cdot \pi_1^{\mathcal{D}} = F \cdot \pi_1^{\mathcal{D}'}$. By axiomatising this situation, as *displayed functors* over functors into the base, such definitions can be stated without equality of objects, with no loss of concision.

▶ **Definition 9** (`disp_functor`)**.** Let $F : \mathcal{C} \to \mathcal{C}'$ be a functor, and $\mathcal{D}$, $\mathcal{D}'$ displayed categories over $\mathcal{C}$ and $\mathcal{C}'$ respectively. A *(displayed) functor G from $\mathcal{D}$ to $\mathcal{D}'$ over F* consists of:

- maps $G_c : \mathcal{D}_c \to \mathcal{D}'_{Fc}$, for each $c : \mathcal{C}$ (which we usually write just as G, omitting c); and
- maps $\mathsf{hom}_f(x, y) \to \mathsf{hom}_{Ff}(Gx, Gy)$, for each $f : c \to c'$ in $\mathcal{C}$;
- satisfying the evident dependent analogues of the usual functor laws.

A displayed functor G over F straightforwardly induces a *total functor* between total categories, written $\int G : \int \mathcal{D} \to \int \mathcal{D}'$, such that $\int G \cdot \pi_1^{\mathcal{D}'} = \pi_1^{\mathcal{D}} \cdot F$. Indeed, displayed functors are precisely equivalent to such functors between total categories. We often therefore call the total functor just G.

Similarly, a functor G over F induces *fibre functors* $G_c : \mathcal{D}_c \to \mathcal{D}'_{Fc}$, for each $c : \mathcal{C}$.

A useful special case is when F is the identity functor of $\mathcal{C}$, in which case we call G just a functor over $\mathcal{C}$; this is precisely equivalent to a functor between the total categories strictly over C in the usual sense.

[2] Our thanks to Mike Shulman and an anonymous referee for pointing out some of these reformulations.

▶ **Definition 10** (`disp_nat_trans`)**.** Let $F, F' : \mathcal{C} \to \mathcal{C}'$ be functors, $\alpha : F \to F'$ a natural transformation, and G and G' displayed functors from $\mathcal{D}$ to $\mathcal{D}'$ over F and F' respectively. A *displayed natural transformation β from G to G' over α* consists of

- for each $c : \mathcal{C}$ and $x : \mathcal{D}_c$, a morphism $\beta_c(x) : \mathsf{hom}_{\alpha(c)}(G(x), G'(x))$
- such that for any $f : \mathcal{C}(c, c')$ and $\bar{f} : \mathsf{hom}_f(x, y)$, $G\bar{f} \cdot \alpha(c') =_* \alpha(c) \cdot G'\bar{f}$.

Just as ordinary functors and natural transormations form a functor category, their displayed versions form a displayed category over the functor category between the bases:

▶ **Definition 11** (`disp_functor_cat`)**.** Given categories $\mathcal{C}$ and $\mathcal{C}'$, and displayed categories $\mathcal{D}$ and $\mathcal{D}'$ over $\mathcal{C}$ and $\mathcal{C}'$ respectively, there is a displayed category $[\mathcal{D}, \mathcal{D}']$ over $[\mathcal{C}, \mathcal{C}']$, defined as follows:

- objects over $F : \mathcal{C} \to \mathcal{C}'$ are displayed functors from $\mathcal{D}$ to $\mathcal{D}'$ over F;
- morphisms over $\alpha : F \to G$ from F' to G' are displayed natural transformations from F' to G' over α;
- composition and identity are given by pointwise composition and identity.

Displayed analogues of usual lemmas on the functor category hold; for instance:

▶ **Lemma 12** (`is_disp_functor_precat_iso_iff_pointwise_iso`)**.** *A displayed natural transformation is an isomorphism in the displayed functor category if and only if it is an isomorphism pointwise.*

We could now go on and define *displayed adjunctions* over adjunctions between the bases, *displayed equivalences* over equivalences of the base, and so on. From these, one gets adjunctions and equivalences, respectively, of total categories. A very useful special case is that of displayed adjunctions and equivalences over the identity in the base, yielding adjunctions and equivalences of total categories leaving the first components of objects untouched.

These definitions are provided in the formalisation; indeed, the original motivation of the present work and formalisation was to have these available, in order to construct an equivalence of univalent categories between CwF-structures and split type-category structures on a fixed base category (cf. the equivalence of types of [3, Construction 19]). However, an account of this is beyond the scope of the present paper.

One may also naturally ask what structure the total collections of displayed categories and natural transformations form. We expect that they should form a bicategory when the base category is held fixed, and more generally a displayed bicategory over the bicategory of categories; but this again is beyond the scope of the present work.

3.3 Constructions on displayed categories

To efficiently construct our remaining key examples, we set up some basic general constructions on displayed categories.

▶ **Definition 13** (`reindex_disp_cat`)**.** Let $\mathcal{D}$ be a displayed cat over $\mathcal{C}$, and $F : \mathcal{C}' \to \mathcal{C}$ a functor. Then $F^*\mathcal{D}$, the *pullback of $\mathcal{D}$ along F*, is the displayed category over $\mathcal{C}'$ defined by

- $(F^*\mathcal{D})_c := \mathcal{D}_{Fc}$
- $\mathsf{hom}_f^{F^*\mathcal{D}}(d, d') := \mathsf{hom}_{Ff}^{\mathcal{D}}(d, d')$

with the evident composition and identities. There is an evident displayed functor $F^*\mathcal{D} \to \mathcal{D}$ over F.

▶ **Example 14** (`disp_cartesian`)**.** Given any categories $\mathcal{C}, \mathcal{C}'$, the *constant displayed category over* $\mathcal{C}$ *with fibre* $\mathcal{C}'$, denoted $\mathsf{const}_{\mathcal{C}}\,\mathcal{C}'$ (or just $\mathsf{const}\,\mathcal{C}'$, when $\mathcal{C}$ is implicit), is the pullback along the unique functor $\mathcal{C} \to 1$ of $\mathcal{C}'$, seen as a displayed category over 1.

There is an evident equivalence from the total category $\int_{\mathcal{C}} \mathsf{const}\,\mathcal{C}'$ to the product $\mathcal{C} \times \mathcal{C}'$, strictly over $\mathcal{C}$.

▶ **Definition 15** (`sigma_disp_cat`)**.** Let $\mathcal{D}$ be a displayed category over $\mathcal{C}$, and $\mathcal{E}$ a displayed category over $\int_{\mathcal{C}} \mathcal{D}$. The Σ-*category* of $\mathcal{E}$ over $\mathcal{D}$, denoted $\sum_{\mathcal{D}} \mathcal{E}$, is the displayed category over $\mathcal{C}$ defined as follows:

- $(\sum_{\mathcal{D}} \mathcal{E})_x := \sum_{y:\mathcal{D}_x} \mathcal{E}_{(x,y)}$
- $\mathsf{hom}_f((y,z),(y',z')) := \sum_{\bar{f}:y \to_f y'} \mathsf{hom}_{(f,\bar{f})}(z,z')$
- operations defined componentwise from those of $\mathcal{D}$ and $\mathcal{E}$.

There is an evident equivalence of total categories $\int_{\left(\int_{\mathcal{C}} \mathcal{D}\right)} \mathcal{E} \to \int_{\mathcal{C}} (\sum_{\mathcal{D}} \mathcal{E})$ over $\mathcal{C}$.

▶ **Example 16** (`disp_arrow`, `disp_domain`, `disp_codomain`)**.** The arrow category has three different displayed incarnations:

1. By $\mathcal{C}^{\to}$, we mean the displayed category over $\mathcal{C} \times \mathcal{C}$ with
   - ${\mathcal{C}^{\to}}_{x,y} := \mathsf{hom}^{\mathcal{C}}(x,y)$
   - $\mathsf{hom}^{\mathcal{C}^{\to}}_{h,k}(f,g) := (f \cdot k = h \cdot g)$, i.e. the proposition that the resulting square commutes.

   As our notation suggests, the total category of this is the usual arrow category of $\mathcal{C}$.
2. Pulling this back along the canonical equivalence $\int_{\mathcal{C}} \mathsf{const}\,\mathcal{C} \to \mathcal{C} \times \mathcal{C}$, and taking the Σ-category of the result, we obtain a displayed category over $\mathcal{C}$ which we denote $-\backslash\mathcal{C}$, since its fibre categories are just the co-slices of $\mathcal{C}$. Its total category is equivalent over $\mathcal{C}$ to $\mathsf{dom} : \mathcal{C}^{\to} \to \mathcal{C}$.
3. If in the previous example, we instead pull back along the equivalence $\int_{\mathcal{C}} \mathsf{const}\,\mathcal{C} \to \mathcal{C} \times \mathcal{C}$ that swaps the two components, we get instead the displayed category of *slices* of $\mathcal{C}$, with total category equivalent over $\mathcal{C}$ to $\mathsf{cod} : \mathcal{C}^{\to} \to \mathcal{C}$.

▶ **Example 17** (`disp_cat_functor_alg`)**.** Suppose $F : \mathcal{C} \to \mathcal{C}$ is an endofunctor. Then F-*algebras* naturally form a displayed category F-Alg over $\mathcal{C}$, with

- $F\text{-}\mathsf{Alg}_c := \mathsf{hom}^{\mathcal{C}}(Fc, c)$
- $\mathsf{hom}_f(\alpha,\beta) := (\alpha \cdot f = Ff \cdot \beta)$, i.e. the proposition that $f : a \to b$ is an algebra homomorphism $(a,\alpha) \to (b,\beta)$.

The total category is the usual category F-Alg. We will sometimes write F-EndAlg to distinguish this from categories of *monad* algebras.

▶ **Example 18** (`disp_cat_monad_alg`)**.** Suppose (T,μ,η) is a monad on $\mathcal{C}$. The full subcategory of F-EndAlg consisting of the *monad* algebras for (T,μ,η) can be seen as a displayed category over F-EndAlg, as in Example 6. Taking the Σ-category of this yields the monad-algebras (T,μ,η)-MonAlg as a displayed category over $\mathcal{C}$.

As usual, we write just T-Alg when there is no risk of confusion.

4 Creation of limits

Creation of limits is our first example of a concept which can be profitably reformulated in terms of displayed categories.

As a property of functors, it is a standard and fruitful tool in category theory. It has however often been viewed with some mistrust for involving equalities of objects: see, for example, [8, Remark 5.3.7].

If formulated instead as a property of displayed categories, it involves no equalities of objects:

▶ **Definition 19** (`creates_limit`)**.** Let $\mathcal{D}$ be a displayed category over $\mathcal{C}$, J a graph, and F a diagram of shape J in $\int\mathcal{D}$. Given a limiting cone λ for the diagram $F \cdot \pi_1 : J \to \mathcal{C}$ in $\mathcal{C}$, with vertex $x : \mathcal{C}$, we say that $\mathcal{D}$ *creates a limit for* F *over* λ if

- there is a unique cone on F over λ; that is, a unique object $d : \mathcal{D}_x$ and family of arrows $\mu_j : \mathsf{hom}_{\lambda_j}(d, \pi_2 F(j))$ such that the pairs (λ_j, μ_j) form a cone on F in $\int\mathcal{D}$;
- and, furthermore, this unique cone $(\lambda_j, \mu_j)_{j:J}$ is limiting.

More generally, we say that $\mathcal{D}$ *creates limits of shape* $\mathcal{J}$ (or *creates small limits*, etc.) if, for any diagram F as above over $\mathcal{J}$ (resp. over any small $\mathcal{J}$), and every limiting cone λ on $F \cdot \pi_1$ in $\mathcal{C}$, $\mathcal{D}$ creates a limit for F over λ.

It is routine to check that this does indeed correspond to the standard notion:

▶ **Proposition 20.** *A displayed category $\mathcal{D}$ over a category $\mathcal{C}$ creates limits of shape $\mathcal{J}$, in our sense, if and only the functor $\pi_1^{\mathcal{D}} : \int\mathcal{D} \to \mathcal{C}$ creates limits in the classical sense.*

It of course follows immediately from this that the displayed definition implies the various standard consequences of creation of limits. In fact, however, the proofs from the displayed definition are at least as direct as the standard proofs; for instance,

▶ **Proposition 21** (`total_limits`, `pr1_preserves_limit`)**.** *Suppose the category $\mathcal{C}$ has limits of shape $\mathcal{J}$, and the displayed category $\mathcal{D}$ over $\mathcal{C}$ creates limits of shape $\mathcal{J}$. Then $\int\mathcal{D}$ has all such limits, and $\pi_1^{\mathcal{D}} : \int\mathcal{D} \to \mathcal{C}$ preserves them.* ◀

Moreover, all the main standard examples of functors that create limits can be seen as the forgetful functors associated to displayed categories.

▶ **Example 22** (`creates_limits_functor_alg`)**.** For any endofunctor $F : \mathcal{C} \to \mathcal{C}$, the displayed category of F-algebras over $\mathcal{C}$ creates all limits. Likewise, for any monad T on $\mathcal{C}$, the displayed category of T-algebras over $\mathcal{C}$ creates all limits.

5 Fibrations

We consider, in this section, three important variations of fibrations of categories: Grothendieck fibrations (and their dual, opfibrations); isofibrations; and discrete fibrations.

We depart from some classical literature in defining fibrations by default to be *cloven* – that is, to include an operation providing all lifts required. (This is not novel: it has been preferred also by other authors, to avoid indiscriminate use of the axiom of choice.) We distinguish the case where liftings are merely known to exist as *weak* fibrations.

5.1 Fibrations and opfibrations

▶ **Definition 23** (`is_cartesian`)**.** Let $\mathcal{D}$ be a displayed category over $\mathcal{C}$. A map $\bar{f} : \mathsf{hom}_f(d', d)$ of $\mathcal{D}$ over $f : c' \to c$ is *cartesian* if for each $g : c'' \to c'$, $d'' : \mathcal{D}_{c''}$, and $\bar{h} : \mathsf{hom}_{f \cdot g}(d'', d)$, there is a unique $\bar{g} : \mathsf{hom}_g(d'', d')$ such that $\bar{g} \cdot \bar{f} =_* \bar{h}$.

▶ **Definition 24** (`cartesian_lift`)**.** Let $\mathcal{D}$ be a displayed category over $\mathcal{C}$. A *cartesian lift* of $f : \mathcal{C}(c', c)$ and $d : \mathcal{D}_c$ consists of an object $d' : \mathcal{D}_{c'}$ and cartesian map $\bar{f} : \mathsf{hom}_f(d', d)$.

▶ **Definition 25** (`cleaving`, `fibration`, `weak_fibration`)**.** A *cleaving* for a displayed category $\mathcal{D}$ over a category $\mathcal{C}$ is a function giving, for each $f : c' \to c$ and $d : \mathcal{D}_c$, a cartesian lift of f and d. A *(cloven) fibration* over $\mathcal{C}$ is a displayed category equipped with a cleaving. A *weak fibration* is a displayed category such that for each such f, d as above, there exists some cartesian lift.

All the above have evident duals: opcartesian maps and lifts, and weak/cloven opfibrations. Again, these all correspond straightforwardly to their classical versions:

▶ **Proposition 26.** *A map in a total category $\int_{\mathcal{C}} \mathcal{D}$ is cartesian in our sense (resp. opcartesian) exactly it is cartesian (opcartesian) with respect to $\pi_1^{\mathcal{D}}$ in the classical sense. A displayed category $\mathcal{D}$ is a cloven (resp. weak) fibration in our sense exactly if $\pi_1^{\mathcal{D}}$ is one in the classical sense (i.e. [8, Def. 5.3.5], read unchanged in the univalent setting).* ◀

As with the standard definition, cartesian lifts are unique up to isomorphism. Proposition 45 below shows that when $\mathcal{D}$ is univalent, they are literally unique.

An important example in our applications of interest is the arrow category:

▶ **Proposition 27** (`cartesian_iff_isPullback`)**.** *For any category $\mathcal{C}$, consider the displayed category $\mathcal{C}/-$ of slices of $\mathcal{C}$, as in Example 16.3 above. An arrow $h : f \to_k g$ in $\mathcal{C}/-$ is cartesian exactly if its associated commuting square is a pullback. The displayed category $\mathcal{C}/-$ is a weak fibration exactly if all pullbacks exist in $\mathcal{C}$, and a (cloven) fibration just if $\mathcal{C}$ has chosen pullbacks.* ◀

Finally, we transfer the definition of *split* fibrations. It seems likely to us that – as with the hom-set condition for categories – split fibrations in the type-theoretic setting should include a setness condition in order to be as useful and well-behaved as classically:

▶ **Definition 28** (`is_split`)**.** Say a fibration $\mathcal{D}$ over $\mathcal{C}$ is *split* if:

- each $\mathcal{D}_c$ is a set; and
- the chosen lifts of idenities are identities, and the chosen lift of any composite is the composite of the individual lifts.

5.2 Isofibrations

▶ **Definition 29** (`iso_disp`)**.** Let $\mathcal{D}$ be a displayed category over $\mathcal{C}$, and $f : c \cong c'$ an isomorphism in $\mathcal{C}$.

A map $\bar{f} : \mathsf{hom}_f(x, y)$ is a *(displayed) isomorphism* if it has a 2-sided inverse, i.e. some $\bar{g} : \mathsf{hom}_{f^{-1}}(y, x)$ such that $\bar{f} \cdot \bar{g} =_* 1_x$ and $\bar{g} \cdot \bar{f} =_* 1_y$ We write $\bar{f} : x \cong_f y$.

As with ordinary isomorphisms, the inverse of a displayed isomorphism is unique.

▶ **Definition 30** (`weak_iso_fibration`, `iso_cleaving`, `iso_fibration`)**.** Let $\mathcal{D}$ be a displayed category over $\mathcal{C}$. Say $\mathcal{D}$ is a *weak isofibration* if for each isomorphism $i : c' \cong c$ in $\mathcal{C}$ and $d : \mathcal{D}_c$, there exists some object $d' : \mathcal{D}_{c'}$ and isomorphism $\bar{i} : d' \cong_i d$. An *iso-cleaving* on $\mathcal{D}$ is a function giving, for each such i, d, some such $d', \bar{i}$. A *(cloven) isofibration* over $\mathcal{C}$ is a displayed category equipped with an iso-cleaving.

▶ **Proposition 31.** *A displayed category is a weak (resp. cloven) isofibration in our sense just if its forgetful functor is one in the classical sense.* ◀

▶ **Example 32** (`iso_cleaving_functor_alg`)**.** The displayed categories of groups, topological spaces, and similar are all naturally isofibrations over set, just as classically. More generally, so are the displayed categories of algebras for endofunctors and monads.

In fact, in the univalent setting, isofibrations often come for free:

▶ **Problem 33** (`iso_cleaving_category`)**.** *Let $\mathcal{D}$ be a displayed category over a univalent category $\mathcal{C}$. Then $\mathcal{D}$ is an isofibration.*

▶ **Construction 34** (for Problem 33)**.** Since $\mathcal{C}$ is univalent, every isomorphism $i : c \cong c'$ is uniquely of the form $\mathsf{idtoiso}(e)$. To give an iso-cleaving on $\mathcal{D}$, it therefore suffices to give, for each $e : c = c'$ and $d : D_{c'}$, some lift $\bar{i} : d' \cong_{\mathsf{idtoiso}\, e} d$. By identity elimination, the case $e := 1_c$ suffices; in this case, we take $d' := d$ and $\bar{i} := 1_d$. ◀

Assuming the univalence axiom, the examples above of grp and top over set therefore come for free. However, we note them separately (and prove them directly, in the formalisation), both to show that they do not require univalence, and to have their action explicitly.

▶ Remark. As the examples given illustrate, most fibrations and isofibrations encountered in nature are categories/functors that arise as the total category/forgetful functor of a displayed category. This, we argue, supports the idea that it is natural to take the displayed-category definitions as basic for developing fibrations and related notions, especially in the type-theoretic setting.

However, not all examples are of this form. For instance, suppose $F : \mathcal{C}' \to \mathcal{C}$ is a functor of small categories that is a complemented inclusion on objects; then the precomposition functor $F^* : \widehat{C} \to \widehat{C'}$ between their presheaf categories is an isofibration. However, in the classical setting, $\widehat{C}$ is not literally the total category of any displayed category over $\widehat{C'}$ (though it is of course isomorphic to one).

5.3 Discrete fibrations

▶ **Definition 35** (`is_discrete_fibration`)**.** Let $\mathcal{D}$ be a displayed category over $\mathcal{C}$. Say that $\mathcal{D}$ is *a discrete fibration* if

- for each $c : \mathcal{C}$, the type $\mathcal{D}_c$ is a set; and
- for any $f : \mathcal{C}(c', c)$ and $d : \mathcal{D}_c$, there is a unique $d' : \mathcal{D}_{c'}$ and $\bar{f} : \mathsf{hom}_f(d', d)$.

These lifts are automatically cartesian; so any discrete fibration is canonically a fibration (`fibration_from_discrete_fibration`), and is moreover split (`is_split_fibration_from_discrete_fibration`).

Thanks to the setness condition, discrete fibrations over a fixed base category $\mathcal{C}$ and displayed functors between them form a category; and, just as classically, we have:

▶ **Problem 36** (`forms_equivalence_disc_fib`)**.** *For any category $\mathcal{C}$, there is a (strong) equivalence of categories between $\widehat{\mathcal{C}}$ and the category of discrete fibrations over $\mathcal{C}$.* ◀

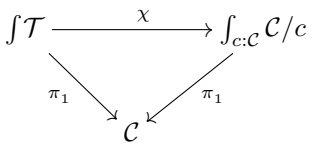
For a presheaf P on $\mathcal{C}$, the classical category of elements of P is the total category of the displayed discrete fibration given by the above equivalence.

6 Comprehension categories

We now turn briefly to *comprehension categories* and *categories with attributes*, just as a glimpse of the applications in semantics of type theory which provided the proximate motivation for the present development.

▶ **Definition 37** (`comprehension_cat_structure`)**.** A *comprehension category* consists of a category $\mathcal{C}$, a fibration $\mathcal{T}$ over $\mathcal{C}$, and a functor $\chi : \mathcal{T} \to \mathcal{C}/-$ over $\mathcal{C}$ (the 'comprehension') preserving cartesian arrows.

Taking total categories and functors, these form a strictly commuting triangle:



As such, this is just a rephrasing of the standard definition [7, Theorem 9.3.4], but avoiding mention of equality on objects.[3]

▶ **Definition 38.** A *split type-category* (aka *category with attributes*) consists of a category $\mathcal{C}$; a presheaf Ty on $\mathcal{C}$; an operation assigning to each $\Gamma : \mathcal{C}$ and $A : \mathsf{Ty}(\Gamma)$ an object and map $\pi_A : \Gamma.A \to A$; and operations giving for each $f : \Gamma' \to \Gamma$ and $A : \mathsf{Ty}(\Gamma)$ a map $f.A$ exhibiting π_{f^*A} as a pullback of π_A.

▶ **Problem 39** ([5, Thm. 2.3])**.** *Any category with attributes induces a comprehension category with the same base.*

▶ **Construction 40** (for Problem 39)**.** The equivalence of Problem 36 turns Ty into a (discrete) fibration. The operations $\Gamma.A$, π_A, and $f.A$ provide the action on objects and arrows of the comprehension functor; while the pullback condition, combined with Proposition 27, ensures that it preserves cartesian maps. ◀

7 Univalence and the Structure Identity Principle

7.1 Displayed univalence

▶ **Definition 41** (`idtoiso_disp`)**.** Let $\mathcal{D}$ be a displayed category over $\mathcal{C}$. Given $c, c' : \mathcal{C}$, $e : c = c'$, $d : \mathcal{D}_c$, $d' : \mathcal{D}_{c'}$, and $e' : d =_e d'$, we write $\mathsf{idtoiso}(e, e') : d \cong_{\mathsf{idtoiso}(e)} d'$ for the canonical displayed isomorphism obtained by identity elimination on e, e'.

Note that we overload the notation $\mathsf{idtoiso}$, using it for both ordinary and displayed categories.

▶ **Definition 42** (`is_univalent_disp`)**.** Let $\mathcal{D}$ be a displayed category over $\mathcal{C}$. Say that $\mathcal{D}$ is *univalent* if for any $c, c' : \mathcal{C}$ and $e : c = c'$ and $d : \mathcal{D}_c$ and $d' : \mathcal{D}_{c'}$, the above map $(d =_e d') \to \mathsf{Iso}_{\mathsf{idtoiso}(e)}(d, d')$ is an equivalence.

To verify univalence of a displayed category, it clearly suffices to prove the condition just in the case where e is reflexivity. But displayed isomorphisms over identities are just isomorphisms in the fibre categories, so we have:

▶ **Proposition 43** (`is_univalent_disp_iff_fibers_are_univalent`)**.** *Let $\mathcal{D}$ be a displayed category over $\mathcal{C}$. Then $\mathcal{D}$ is univalent exactly if each of its fibre categories is univalent.*

The key practical application of displayed univalence is in proving that complex categories built up using displayed categories are univalent:

▶ **Theorem 44** (`is_univalent_total_category`)**.** *Let $\mathcal{C}$ be a univalent category, and let $\mathcal{D}$ be a univalent displayed category over $\mathcal{C}$. Then the total category $\int \mathcal{D}$ is univalent.* ◀

However, displayed univalence is a meaningful notion even when the base is not known to be univalent; one has, for instance:

[3] Jacobs' original formulation [6, Definition 4.1] is slightly different.

▶ **Proposition 45** (`isaprop_cartesian_lifts`, `univalent_fibration_is_cloven`)**.** *Let $\mathcal{D}$ be a* univalent *displayed category over $\mathcal{C}$. For any $f : c' \to c$ and $d : \mathcal{D}_c$, if a cartesian lift $(d', \bar{f})$ of f and d exists, then it is unique; that is, the type of cartesian lifts is a proposition. More generally, if $\mathcal{D}$ is a weak (iso-)fibration, then it possesses a unique (iso-)cleaving.*

Proof. The usual classical argument shows that cartesian lifts are unique up to isomorphism. By univalence of $\mathcal{D}$, it follows that they are literally unique.

It follows that the type of (iso-)cleavings of $\mathcal{D}$ is a proposition; and that whenever a suitable lift is known to exist, one can be chosen. Putting these together, the proposition follows. ◀

Similarly, as for ordinary categories, univalence bounds the h-level of the types of objects:

▶ **Proposition 46** (`univalent_disp_cat_has_groupoid_obs`)**.** *Let $\mathcal{D}$ be a univalent displayed category over $\mathcal{C}$. Then for each $c : \mathcal{C}$, the type of objects $\mathcal{D}_c$ is a 1-type.* ◀

7.2 Structure Identity Principle

Theorem 44 generalizes an early-noted consequence of univalence, the so-called *structure identity principle*, as formulated by Aczel. We recall that here for comparison.

▶ **Definition 47** ([9, Def. 9.8.1])**.** A *standard notion of structure* on a category $\mathcal{C}$ consists of:

1. for each $c : \mathcal{C}$, a type $P(c)$;
2. for each $c, c' : \mathcal{C}$ and $\alpha : P(c)$ and $\beta : P(c')$ and $f : \mathcal{C}(c, c')$, a proposition $H_{\alpha,\beta}(f)$;
3. such that H is suitably closed under composition and identity; and
4. for each $c : \mathcal{C}$, the preorder on $P(c)$ defined by setting $\alpha \leq \alpha'$ if $H_{\alpha,\alpha'}(1_c)$ is a poset.

Items 1–3 can immediately be read as providing an associated displayed category over $\mathcal{C}$ (`disp_cat_from_SIP_data`), whose displayed hom-sets are propositions. The category of (P, H)*-structures*, as defined in [9], is precisely the total category of this displayed category.

With a little thought, item 4 can then be seen as saying that this displayed category is univalent (`is_univalent_disp_from_SIP_data`). Theorem 44 then immediately implies:

▶ **Corollary 48** ([9, Theorem 9.8.2])**.** *Given a standard notion of structure (P, H) on $\mathcal{C}$, if $\mathcal{C}$ is univalent, then so is the category of (P, H)-structures on $\mathcal{C}$.*

▶ **Example 49** (`is_univalent_disp_functor_alg`)**.** The displayed categories of algebras for an endofunctor or monad (Examples 17, 18) arise from standard notions of structure, and so are univalent.

▶ Remark. The displayed categories arising in this way – univalent, and with all hom-sets propositions – may also be compared to the *amnestic* concrete categories of [1, Def. 3.27(4)].

8 Conclusions

We have introduced displayed categories, and set up their basic theory, along with key examples and applications.

The applications fall into two main groups:

- rephrasing classical definitions to avoid referring to equality of objects;
- allowing categories of multi-component structures, and maps between such categories, to be constucted and reasoned about in a modular, stage-by-stage fashion.

In this paper, we have focused more on the former – for instance, the use of displayed categories as a basis for the development of fibrations in the type-theoretic setting.

We have seen less of the latter, since it is typically tied to specific more involved applications. However, in our own further work (for instance, on the structures considered in [3]), we have found this at least as significant as a payoff of the present work.

Theorem 44, giving univalence of the total category, is especially valuable. Naïve approaches to proving univalence quickly become quite cumbersome even for categories of only moderately complex structures, such as groups. The issue is that identities between such structures translate to a tuples of identities between the components, where the identities of later components are usually heterogeneous, involving accumulated transports along the identities between earlier components.

The displayed-category approach avoids this; one need only work 'fibrewise', over each component in turn. All the necessary wrangling of transports is dealt with once and for all in the proof of Theorem 44.

An instance of this is the proof of univalence of the category of CwF-structures over a fixed univalent base category. Details are beyond the scope of the present article; but it is available in the formalisation as `is_univalent_term_fun_structure`.

Further work

In the present article and formalisation, we have explored only the basic theory and applications of displayed categories. There are many clear directions for further work:

- In [3], we have started a project of giving careful comparisons between the various categorical structures used for semantics of type theory. We touched on this project in Section 6. In forthcoming work, we plan to give full comparisons between categories of such structures, including comprehension categories, type-categories (not necessarily split), and categories with display maps.
- The material on creation of limits in Section 4 should be generalised to a more permissive notion of *displayed limits*, to cover a broader range of examples.
- In the formalisation (though not the article) we study displayed adjunctions and equivalences over a fixed base, and show that these induce adjunctions and equivalences between total categories and fibre categories. This should be generalised to displayed adjunctions/equivalences over adjunctions/equivalences in the base.
- Generally, one should be able to assemble displayed categories into a displayed bicategory over the bicategory of categories. Of course, this would require defining displayed bicategories, and developing the basic theory of bicategories in the type-theoretic setting.
- Displayed categories should also be vieweable as forming some 2-dimensional analogue of a comprehension category, with displayed categories being the 'dependent types' over a base category 'context'. This would provide a new potential guiding example for the 'directed type theory' that various authors have started to explore in recent work.

Acknowledgements. We would like to thank Mike Shulman and the participants of the Stockholm Logic Seminar for very helpful feedback on this article and the work it reports on.

We want to thank the actors of the EUTypes COST Action CA 15123 for providing funding for a research visit in the course of which this article was authored.

References

1. Jiří Adámek, Horst Herrlich, and George E. Strecker. *Abstract and concrete categories: The joy of cats*. Pure and Applied Mathematics (New York). John Wiley & Sons, Inc., New York, 1990. URL: http://www.tac.mta.ca/tac/reprints/articles/17/tr17abs.html.
2. Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. Univalent categories and the Rezk completion. *Mathematical Structures in Computer Science*, 25:1010–1039, 2015. arXiv:1303.0584, doi:10.1017/S0960129514000486.
3. Benedikt Ahrens, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. Categorical structures for type theory in univalent foundations, 2017. To be published in proceedings of Computer Science Logic (CSL) 2017. arXiv:1705.04310.
4. Jean Bénabou. Distributors at work. Notes by Thomas Streicher from lectures given at TU Darmstadt, 2000. URL: http://www.mathematik.tu-darmstadt.de/~streicher/FIBR/DiWo.pdf.
5. Javier Blanco. Relating categorical approaches to type theory, 1991. Master thesis, Univ. Nijmegen.
6. Bart Jacobs. Comprehension categories and the semantics of type dependency. *Theor. Comput. Sci.*, 107(2):169–207, 1993. doi:10.1016/0304-3975(93)90169-T.
7. Bart Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1999.
8. Tom Leinster. *Basic Category Theory*, volume 143 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 2014.
9. The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book, Institute for Advanced Study, 2013.
10. Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. *UniMath*: Univalent Mathematics. Available at https://github.com/UniMath.

The Confluent Terminating Context-Free Substitutive Rewriting System for the lambda-Calculus with Surjective Pairing and Terminal Type

Yohji Akama

Department of Mathematics, Tohoku University, Sendai, Miyagi, Japan

Abstract

For the lambda-calculus with surjective pairing and terminal type, Curien and Di Cosmo, inspired by Knuth-Bendix completion, introduced a confluent rewriting system of the naive rewriting system. Their system is a confluent (CR) rewriting system stable under contexts. They left the strong normalization (SN) of their rewriting system open. By Girard's reducibility method with restricting reducibility theorem, we prove SN of their rewriting, and SN of the extensions by polymorphism and (terminal types caused by parametric polymorphism). We extend their system by sum types and eta-like reductions, and prove the SN. We compare their system to type-directed expansions.

1998 ACM Subject Classification F4.1 Mathematical Logic, F4.2 Grammars and Other Rewriting Systems

Keywords and phrases reducibility method, restricted reducibility theorem, sum type, type-directed expansion, strong normalization

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.6

1 Introduction

We recall the equational theory $\lambda\beta\eta\pi*$ from [9]. Types are built up from the distinguished type constant $\top$, and type variables by means of the product type $\varphi \times \psi$ and the function type $\varphi \to \psi$. Terms are built up from the distinguished term constant $*^{\top}$ and term variables $x^{\varphi}, y^{\varphi}, \dots, x^{\psi}, y^{\psi}, \dots$ by means of λ-abstraction $(\lambda x^{\varphi}.\, t^{\psi})^{\varphi\to\psi}$, term application $(u^{\varphi\to\psi} v^{\varphi})^{\psi}$, pairing $\langle u^{\varphi}, v^{\psi}\rangle^{\varphi\times\psi}$, left-projection $(\pi_1 t^{\varphi\times\psi})^{\varphi}$, right-projection $(\pi_2 t^{\varphi\times\psi})^{\psi}$, where the superscript represents the type. The superscript is often omitted. The set of free variables of a term t is denoted by $\mathrm{FV}(t)$. The equational theory $\lambda\beta\eta\pi*$ consists of the following axioms:

$$(\beta) \quad (\lambda x.\, u)v = u[x := v].$$

$$(\pi_1) \quad \pi_1\langle u, v\rangle = u. \qquad (\pi_2) \quad \pi_2\langle u, v\rangle = v.$$

$$(\eta) \quad \lambda x.\, tx = t, \quad (x \in \mathrm{FV}(t).)$$

$$(SP) \quad \langle \pi_1 u, \pi_2 u\rangle = u.$$

$$(c) \quad s^{\top} = *^{\top}.$$

By the last equality, the type $\top$ corresponds to the singleton. The singleton does to the terminal object of a cartesian closed category. So $\top$ is called the *terminal type*.

A confluent (CR for short) and weakly normalizable (WN for short) reduction system generating this equational theory $\lambda\beta\eta\pi*$ is important in relation to the coherence problem of cartesian closed categories [28, 29]. By orienting the axioms $(\beta), (\pi_1), (\pi_2), (\eta), (SP)$ left

to right, we obtain rewriting rule schemata. Let (T) be a rewrite rule schema $s^{\top} \to *^{\top}$ $(s^{\top} \not\equiv *^{\top})$. Here for terms t and s, we write $t \equiv s$, provided that by renaming bound variables, t becomes identical to s. Let $\to$ be the closure of these rewriting rule schemata by contexts. By abuse of notation, we write $\lambda\beta\eta\pi*$ for a so-obtained rewriting system. The reverse of $\to$ is denoted by $\leftarrow$. $\overset{*}{\to}$ is the reflexive, transitive closure of $\to$. The rewriting system $\lambda\beta\eta\pi*$ is *not* CR, as follows: In each line of the following, x and y are variables, and it is not the case that there is a term t_0 such that $t_1 \overset{*}{\to} t_0 \overset{*}{\leftarrow} t_2$:

$$\begin{aligned} y^{\varphi\to\top} &\leftarrow \lambda x.\,(yx)^{\top} \to \lambda x.\,*^{\top}, \\ x &\leftarrow \langle (\pi_1 x)^{\top},\, (\pi_2 x)^{\top}\rangle \to\to \langle *^{\top},\, *^{\top}\rangle, \\ \lambda x^{\top}.\,y* &\leftarrow \lambda x^{\top}.\,yx^{\top} \to y^{\top\to\varphi}, \\ \langle \pi_1 x,\, *\rangle &\leftarrow \langle (\pi_1 x)^{\varphi},\, (\pi_2 x)^{\top}\rangle \to x^{\varphi\times\top}, \\ \langle *,\, \pi_2 x\rangle &\leftarrow \langle (\pi_1 x)^{\top},\, (\pi_2 x)^{\varphi}\rangle \to x^{\top\times\varphi}. \end{aligned} \tag{1}$$

If we omit the rewrite rule schemata (T), then the resulting rewrite relation $\to_{\beta\eta\pi\pi_1\pi_2 SP}$ is CR [31]. In the type-free setting, $\to_{\beta SP}$ is not CR [23]. The decidability of the equational theory $\lambda\beta\eta\pi*$ follows from

- Sarkar's algorithm. For an extension of the well-known LF type theory with dependent pair and unit types, Sarkar [32] provided an algorithm that decides type checking and he proved the existence of canonical forms, by using standard techniques introduced by Harper and Pfenning [16].
- A translation that incorporates type-directed expansions by type-indexed functions on terms. The translation reduces the decidability of the equational theory $\lambda\beta\eta\pi*$ to that of the corresponding intensional equational theory, as in [14, 38]. This idea, however, does not yield a decision procedure for the polymorphic equational theory.

1.1 Why do we insist on η-reduction instead of type-directed η-expansion?

For *Unifying Theory of Dependent Types* (UTT for short) (Luo [26]), Goguen [13] defined the *typed operational semantics* (TOS for short). By employing the TOS, he investigated various decidability properties of UTT. Here UTT is Martin-Löf's Logical Framework extended by a general mechanism for inductive types, a predicative universe and an impredicative universe of propositions. Goguen's TOS defines a reduction to normal form for terms which are well-typed in UTT. "Since his approach is based on η-reduction instead of η-expansion, it is not clear whether it scales to a unit type with extensional equality." ([1]). The unit type is exactly the terminal type, and is related to types for enumeration sets and types for proof irrelevance [2].

The TOS is an intermediate induction principle for reducibility proofs. By using Curien-Di Cosmo's idea, we could hopefully formulate the reduction system for UTT+unit type, so that the TOS for UTT+unit type can be defined, where all terms of unit type reduce to the unique inhabitant, and then used to show termination.

We conjecture that for each term t, the minimum length of the normalization sequence from t with respect to $(\lambda\beta\eta\pi*)'$ is smaller than the minimum length of the normalization sequence from t with respect to $\beta\pi_1\pi_2 T$-reduction union the type-directed ηSP-expansions. If so, a type-checker of dependent type theories perhaps run faster by using $(\lambda\beta\eta\pi*)'$ instead of type-directed expansions, since a type-checker tests term equivalence.

1.2 Curien and Di Cosmo's rewriting system based on η-reduction

For the equational theory $\lambda\beta\eta\pi*$, Curien and Di Cosmo, inspired by completion of TRS, introduced a rewriting system $(\lambda\beta\eta\pi*)'$ in [9]. First they inductively defined the types *"isomorphic to"* the terminal type $\top$ and the *canonical terms* of such types.

▶ **Definition 1** ($(\lambda\beta\eta\pi*)'$).

- $\top$ is "isomorphic to" $\top$ and the canonical term of $\top$ is $*^{\top}$.
- Suppose φ is a type and τ is a type "isomorphic to" $\top$. Then the type $\varphi \to \tau$ is "isomorphic to" $\top$ and the canonical term $*^{\varphi\to\tau}$ of $\varphi \to \tau$ is $\lambda x^{\varphi}.*^{\tau}$.
- If each type τ_i is "isomorphic to" $\top$ $(i = 1, 2)$, then the type $\tau_1 \times \tau_2$ is "isomorphic to" $\top$ and the canonical term $*^{\tau_1\times\tau_2}$ of $\tau_1 \times \tau_2$ is $\langle *^{\tau_1}, *^{\tau_2}\rangle$.

The set of types "isomorphic to" $\top$ is denoted by $Iso(\top)$.

Whenever we write $*^{\varphi}$, we tacitly assume $\varphi \in Iso(\top)$.

The rewrite relation $\to$ of the rewriting system $(\lambda\beta\eta\pi*)'$ is defined by the rewrite rule schemata obtained from the first five equational axioms $(\beta), (\pi_1), (\pi_2), (\eta)$, and (SP) of $\lambda\beta\eta\pi*$ by orienting left to right, and the following four rewrite rule schemata:

(g)	$u^{\tau} \to *^{\tau}$,	(u is not canonical.)
(η_{top})	$\lambda x^{\tau}.\, t*^{\tau} \to t$,	($x \notin \mathrm{FV}(t)$.)
(SP_{top1})	$\langle \pi_1 u, *^{\tau}\rangle \to u$,	(u has type $\varphi \times \tau$.)
(SP_{top2})	$\langle *^{\tau}, \pi_2 u\rangle \to u$,	(u has type $\tau \times \psi$.)

g stands "gentop." In [9], Curien and Di Cosmo proved that the rewriting system $(\lambda\beta\eta\pi*)'$ is CR and WN, by using an ingenuous lemma for abstract reduction system. $(\lambda\beta\eta\pi*)'$ is non-left-linear and has a rewrite rule schema with side conditions. We cannot apply criteria for CR of left-linear (higher-order) term rewriting system based on closed condition of (parallel) critical pairs (e.g., [35, 36]). $\beta\eta\eta_{top}T$-reduction is the *triangulation* [37] of $\beta\eta T$-reduction, and thus CR by [37, Corollary 2.6]. However, $(\lambda\beta\eta\pi*)'$ is not a triangulation of the rewriting system $\lambda\beta\eta\pi*$. As we see (1), g-rule schema rewrites the one-step reduct $u^{\top\times\top}$ of $\langle \pi_1 u, \pi_2 u\rangle$ to the *two-step* reduct of $\langle \pi_1 u, \pi_2 u\rangle$. This does not fit to the definition of the triangulation.

1.3 Curien and Di Cosmo's attempted to prove SN of their rewriting system

All variations (e.g., Girard [12], Blanqui (computability closure [6])) of Tait's reducibility method require to show a key statement like *"if $v[x := u]$ is reducible for all reducible u, then $\lambda x.\, v$ is reducible,"* where we say a term t of type $\varphi \to \psi$ is reducible if for all reducible term u of type φ, tu is reducible. An auxiliary property which is available is that, a term tu is reducible, as soon as s is reducible for all reducts s of tu, for example, in [12]. So the proof of the key statement amounts to the proof that all reducts of $(\lambda x.\, v)u$ are reducible. Now, if $v \equiv (v'*)$ with $x \notin \mathrm{FV}(v')$, then the rule schema (η_{top}) can rewrite $(\lambda x.\, v)u$ to $(v'u)$ which is not $v[x := u] \equiv v$, and we do not know if $(v'u)$ *is reducible.*

Curien and Di Cosmo proved SN of g-normal forms without the second-order β-rewrite rule schema (β^2) but with the second-order η-rewrite rule schema (η^2), and SN of all the terms with both (β^2) and (η^2) but without (η_{top}), (SP_{top1}) and (SP_{top2}). But these do not lead to SN of all terms for the full reduction.

1.4 Our SN proof of Curien-Di Cosmo's reduction systems

To carry out Girard's reducibility [12] for $(\lambda\beta\eta\pi*)'$,

1. We prove

   (CR0) "every canonical term is reducible,"

   besides Girard's three properties (CR1) ("the reducibility implies SN."), (CR2) ("the reducibility is closed under $\to$."), and (CR3).
2. In the reducibility method for $(\lambda\beta\eta\pi*)'$, as a sufficient condition for $\lambda x.\, v$ to be reducible, we consider the conjunction of (a) $v[x := u]$ is reducible for all reducible u and (b) v' is reducible whenever $v \equiv v'*$.
3. We will restrict the reducibility theorem "all terms substituted by reducible terms are reducible" to $*$-*free* terms.

▶ **Definition 2.** Let t be a term of $(\lambda\beta\eta\pi*)'$.

1. t is called $*$-*free*, if the term constant $*^{\top}$ does not occur in t.
2. Let $\tilde{t}$ be a $*$-free term such that all the occurrences of $*^{\top}$ in t is replaced by variables $x^{\top}$.

None of the problematic rewrite rule schemata (η_{top}), (SP_{top1}), and (SP_{top2}) applies for a $*$-free term. So we can prove the restricted reducibility theorem, as usual, by induction on t, but we use

▶ **Lemma 3.** *Let t be a term of $(\lambda\beta\eta\pi*)'$. If $\tilde{t}$ is reducible, so is t.*

Proof. By $\tilde{t} \xrightarrow{*}_T t$ and (CR2). ◀

From the restricted reducibility theorem, we derive the reducibility of *all* terms, again by Lemma 3. The condition (CR1) establishes SN of $(\lambda\beta\eta\pi*)'$. Thus, we reduced SN of the terms to SN of the same rewriting relation for the $*$-free terms. This kind of trick to restrict terms is also found in the normalization by evaluation for a dependent type theory with enumeration sets and types for proof irrelevance ([2]).

The rest of paper is organized as follows: In Section 2, we prove SN of $(\lambda\beta\eta\pi*)'$. In Section 3, we prove (1) SN of $\left(\lambda^2\beta\eta\pi*\right)'$, the extension by the second-order $\beta\eta$-rewriting and (2) SN of $\left(\lambda^2\beta\eta\pi*\right)''$, the extension by the second-order $\beta\eta$-rewriting where we also consider terminal types caused by parametric polymorphism [17]. In Section 4, we show the worst-case derivational complexity of $(\lambda\beta\eta\pi*)'$ is smaller than that of so-called type-direwected expansions. For type-directed expansions, see [28, 29, 15, 3, 8, 10, 20, 25], to cite a few. In the appendix, we prove SN of $\left(\lambda^{\top,\to,\times,+}\right)'$, the extension of $(\lambda\beta\eta\pi*)'$ by sum types with weak extensionality.

2 SN proof of $(\lambda\beta\eta\pi*)'$, by restricted reducibility theorem

In our SN proofs, we will use a *well-founded induction on a well-founded relation*. A *well-founded* relation is, by definition, $\mathcal{A} = (A, \succ)$ such that $\emptyset \neq \succ \subseteq A \times A$ and there is no infinite chain $a \succ a' \succ a'' \succ \cdots$. The *well-founded induction* on a well-founded relation $\mathcal{A} = (A, \succ)$ is, by definition,

$$\mathrm{WFI}(\mathcal{A}): \quad \forall P \subseteq A \left[\forall x \in A\big(\forall x' \left(x \succ x' \Rightarrow x' \in P\right) \implies x \in P\big) \implies \forall x \in A(x \in P)\right].$$

We call the subformula $\forall x' \left(x \succ x' \Rightarrow x' \in P\right)$ the *WF induction hypothesis*. For $n \geq 1$ well-founded relations $\mathcal{A}_i = (A_i, \succ_i)$ $(i = 1, \ldots, n)$, we define a binary relation $\mathcal{A}_1 \# \cdots \# \mathcal{A}_n = (A_1 \times \cdots \times A_n,\ \succ_1 \# \cdots \# \succ_n)$ by: $(x_1, \ldots, x_n)\ (\succ_1 \# \cdots \# \succ_n)\ (y_1, \ldots, y_n)$, if there exists i such that $x_i \succ_i y_i$ but $x_j = y_j$ $(j \neq i)$. Then $\mathcal{A}_1 \# \cdots \# \mathcal{A}_n$ is a well-founded relation.

If the redex of $t \to t'$ is Δ, we write $t \overset{\Delta}{\to} t'$. Below, "$\subseteq$" reads "is a subterm occurrence of."

▶ **Definition 4** (Neutral [12]). A term is called *neutral* if it is not of the form $\langle u, v\rangle$ or $\lambda x.\, v$.

By an *atomic type*, we mean the distinguished type constant $\top$ or a type variable.

▶ **Definition 5** (Reducibility [12]).
(a) A term of atomic type is *reducible*, if the term is SN.
$(\times)$ A term $t^{\varphi\times\psi}$ is *reducible*, if so are $(\pi_1 t)^{\varphi}$ and $(\pi_2 t)^{\psi}$.
$(\to)$ A term $t^{\varphi\to\psi}$ is *reducible*, if for any reducible term u^{φ}, $(tu)^{\psi}$ is reducible.

We state and prove four properties (CR0), (CR1), (CR2) and (CR3) of the reducibility, where the last three are the same as those Girard proved [12] for $\beta\pi_1\pi_2$-reduction.

▶ **Lemma 6.** *Let t^{φ} be a term.*
(CR0) *If t is canonical, then t is reducible.*
(CR1) *If t is reducible, then t is SN.*
(CR2) *if t is reducible and $t \to t'$, then t' is reducible.*
(CR3) *if t is neutral, and t' is reducible whenever $t \to t'$, then t is reducible.*
To prove Lemma 6, we first note the following:

▶ **Lemma 7.** *By (CR0) and (CR3), we have*
(CR4) *If t is a variable, then t is reducible.*

Proof. Let $t \to t'$. Then t' is canonical. By (CR0), t' is reducible. By (CR3), t is too. ◀

Proof of Lemma 6. By induction on φ.

φ is atomic.
(CR0) t is $*^{\top}$, and SN. So t is reducible. (CR1) is clear. (CR2) As t is SN, so is every reduct t' of t. (CR3) If all reducts are SN, then it is SN.

$\varphi = \varphi_1 \times \varphi_2$.
(CR0) As $*^{\varphi_1\times\varphi_2}$ is a normal form $\langle *^{\varphi_1}, *^{\varphi_2}\rangle$, the reduct of $\pi_i *^{\varphi_1\times\varphi_2}$ is $*^{\varphi_i}$, which is reducible by induction hypothesis (CR0). By induction hypothesis (CR3) for φ_i, $\pi_i *^{\varphi_1\times\varphi_2}$ is reducible. Hence $*^{\varphi_1\times\varphi_2}$ is reducible.
(CR1) Suppose that t is reducible. Then $\pi_i t$ is reducible. By induction hypothesis (CR1) for φ_i, $\pi_i t$ is SN. So t is SN.
(CR2) If $t \to t'$, then $\pi_i t \to \pi_i t'$. As t is reducible by hypothesis, so are $\pi_i t$. By induction hypothesis (CR2) for φ_i, $\pi_i t'$ is reducible, and so t' is reducible.
(CR3) Let $\pi_i t \overset{\Delta}{\to} s$. We have two cases.
1. $\Delta \equiv \pi_i t$: Then $\varphi_i \in \mathit{Iso}(\top)$ and $s \equiv *^{\varphi_i}$, because t is neutral. By induction hypothesis (CR0) for φ_i, s is reducible.
2. $\Delta \not\equiv \pi_i t$: Then $s \equiv \pi_i t'$ for some t' such that $t \to t'$. By the hypothesis, t' is reducible. So s is reducible. $\pi_i t$ is neutral, and all the terms s with $\pi_i t \to s$ are reducible. By induction hypothesis (CR3) for φ_i, $\pi_i t$ is reducible. Hence t is reducible.

$\varphi = \varphi_1 \to \varphi_2$.
(CR0) Let u be a reducible term of type φ_1. By induction hypothesis (CR1) for φ_1, u is SN. So we can use $\mathrm{WFI}\left(\left(\{u^{\varphi_1} \mid u^{\varphi_1} \text{ is reducible}\},\ \to\right)\right)$ where $\to$ is the rewrite relation. We will verify that $*^{\varphi_1\to\varphi_2} u$ is reducible. Suppose $*^{\varphi_1\to\varphi_2} u \overset{\Delta}{\to} s$. As $*^{\varphi_1\to\varphi_2}$ is in normal form, we have two cases.

1. $\Delta \equiv *^{\varphi_1 \to \varphi_2} u$: Then $s \equiv *^{\varphi_2}$ is reducible by induction hypothesis (CR0) for φ_2.
2. Otherwise, $s \equiv *^{\varphi_1 \to \varphi_2} u'$ with $u \to u'$. Then u' is reducible by induction hypothesis (CR2) for φ_1. So, by the WF induction hypothesis, $s \equiv *^{\varphi_1 \to \varphi_2} u'$ is reducible.

In any case, the neutral term $*^{\varphi_1 \to \varphi_2} u$ rewrites to reducible terms only. By induction hypothesis (CR3) for φ_2, $*^{\varphi_1 \to \varphi_2} u$ is reducible. So $*^{\varphi_1 \to \varphi_2}$ is reducible.

(CR1) By induction hypothesis (CR4), a variable x^{φ_1} is reducible. So tx is reducible. Hence t is SN.

(CR2) Let u be a reducible term of type φ_1. Then tu is reducible and $tu \to t'u$. By the induction hypothesis (CR2) for φ_2, $t'u$ is reducible. So t' is reducible.

(CR3) Assume t be neutral and suppose all the t' with $t \to t'$ are reducible. Let u be a reducible term of type φ_1. By induction hypothesis (CR1) for φ_1, u is SN. So by $\mathrm{WFI}\left(\left(\left\{u^{\varphi_1} \mid u^{\varphi_1} \text{ is reducible}\right\}, \to\right)\right)$, we will verify that tu is reducible.

Suppose $tu \xrightarrow{\Delta} s$. We will show that s is reducible. As t is neutral, we have three cases.

1. $\Delta \equiv tu$: Then, $s \equiv *^{\varphi_2}$ is reducible, by induction hypothesis (CR0) for φ_2.
2. $\Delta \subseteq t$: Then, $s \equiv t'u$ with $t \to t'$. $t'u$ is reducible, because t' is by the assumption,
3. Otherwise, $s \equiv tu'$ with $u \to u'$. Then, u' is reducible by induction hypothesis (CR2) for φ_1. So, by the WF induction hypothesis, tu' is reducible.

In any case, the neutral term tu rewrites to reducible terms only. By induction hypothesis (CR3) for φ_2, tu is reducible. So t is reducible. This completes the proof of Lemma 6. ◀

For pairings and λ-abstractions to be reducible, we consider a sufficient condition stronger than that used in standard reducibility methods (e.g., [12]). In view of the rules (SP_{top1}), (SP_{top2}), and (η_{top}), we newly consider (1(b)), (1(c)) and (2(b)).

▶ **Lemma 8.**

1. *Let u^{φ}, v^{ψ} be any terms. $\langle u^{\varphi}, v^{\psi} \rangle$ is reducible, provided that*
   - **(a)** *u and v are both reducible;*
   - **(b)** *if $u \equiv \pi_1 w$ and $v \equiv *^{\psi}$, then w is reducible; and*
   - **(c)** *if $v \equiv \pi_2 w$ and $u \equiv *^{\varphi}$, then w is reducible.*
2. *Let v^{ψ} be any term. $\lambda x^{\varphi}.\, v^{\psi}$ is reducible, provided that*
   - **(a)** *$v^{\psi}[x^{\varphi} := u^{\varphi}]$ is reducible for every reducible, possibly non-$*$-free term u^{φ}; and*
   - **(b)** *if $v \equiv w^{\varphi \to \psi} *^{\varphi}$ and $x^{\varphi} \notin \mathrm{FV}(w^{\varphi \to \psi})$, then $w^{\varphi \to \psi}$ is reducible.*

Proof. (1) By the premise and (CR1), u and v are both SN. We can use

$$\mathrm{WFI}\left(\left(\left\{u^{\varphi} \mid u^{\varphi} \text{ is reducible}\right\}, \to\right) \# \left(\left\{v^{\psi} \mid v^{\psi} \text{ is reducible}\right\}, \to\right)\right) \tag{2}$$

where $\to$ is the rewrite relation. We will verify that $\pi_1\langle u, v\rangle$ is reducible. Let $\pi_1\langle u, v\rangle \xrightarrow{\Delta} s$. We will prove that s is reducible, by case analysis. We will exhaust the positions of the redexes Δ in $\pi_1\langle u, v\rangle$ from left to right, and the rewrite rule schemata of $\xrightarrow{\Delta}$. We have eight cases.

1. $\Delta \equiv \pi_1\langle u, v\rangle$ is a redex of the rewrite rule (g) and $s \equiv *^{\varphi}$: Then s is reducible by (CR0).
2. $\Delta \equiv \pi_1\langle u, v\rangle$ is a redex of the rewrite rule (π_1) and $s \equiv u$: Then s is reducible by the hypothesis (1(a)).
3. $\Delta \equiv \langle u, v\rangle$ is a redex of (g) and $s \equiv \pi_1(*^{\varphi \times \psi})$: Then $*^{\varphi \times \psi}$ is reducible by (CR0). By the definition of the reducibility for the product type, $s \equiv \pi_1(*^{\varphi \times \psi})$ is reducible.
4. $\Delta \equiv \langle u, v\rangle$ is a redex of (SP) and $s \equiv \pi_1 w$: Then $u \equiv \pi_1 w$ and $v \equiv \pi_2 w$. $s \equiv \pi_1 w$ is reducible by the hypothesis (1(a)).
5. $\Delta \equiv \langle u, v\rangle$ is a redex of (SP_{top1}) and $s \equiv \pi_1 w$: Then $u \equiv \pi_1 w$ and $v \equiv *^{\psi}$. $s \equiv \pi_1 w$ is reducible by the hypothesis (1(b)).

6. $\Delta \equiv \langle u, v\rangle$ is a redex of (SP_{top2}) and $s \equiv \pi_1 w$: Then $v \equiv \pi_2 w$ and $u \equiv *^{\varphi}$. $s \equiv \pi_1 w$ is reducible by the hypothesis (1(c)).
7. $\Delta \subseteq u$: Then $s \equiv \pi_1\langle u', v\rangle$ with $u \to u'$. u' is reducible by (1(a)) and (CR2). By the WF induction hypothesis, $s \equiv \pi_1\langle u', v\rangle$ is reducible.
8. $\Delta \subseteq v$: Then $s \equiv \pi_1\langle u, v'\rangle$ with $v \to v'$. v' is reducible by (1(a)) and (CR2). By the WF induction hypothesis, $s \equiv \pi_1\langle u, v'\rangle$ is reducible.

In every case, the neutral term $\pi_1\langle u, v\rangle$ rewrites to reducible terms only, and by (CR3), $\pi_1\langle u, v\rangle$ is reducible. We can similarly prove that $\pi_2\langle u, v\rangle$ is reducible. So $\langle u, v\rangle$ is reducible.

(2) By (CR4), x^{φ} is reducible. So v^{ψ} is, by the premise. Let u^{φ} be a reducible, possibly non-$*$-free term. By (CR1), both of u, v are SN. By (2), we will verify that $(\lambda x.\, v)u$ is reducible. Assume $(\lambda x.\, v)u \overset{\Delta}{\to} s$. We will exhaust the positions of the redex Δ in $(\lambda x.\, v)u$ from left to right, and the rewrite rule schemata of $\overset{\Delta}{\to}$. Then we have seven cases:

1. $\Delta \equiv (\lambda x.\, v)u$ is a redex of (g) and $s \equiv *^{\psi}$: Then s is reducible by (CR0).
2. $\Delta \equiv (\lambda x.\, v)u$ is a redex of (β) and $s \equiv v[x := u]$: Then s is reducible by hypothesis (2(a)).
3. $\Delta \equiv \lambda x.\, v$ is a redex of (g) and $s \equiv *^{\varphi\to\psi} u$: As $*^{\varphi\to\psi}$ is reducible by (CR0), so is s.
4. $\Delta \equiv \lambda x.\, v$ is a redex of (η) and $s \equiv v[x := u]$: Then, this case is case 2.
5. $\Delta \equiv \lambda x.\, v$ is a redex of (η_{top}) and $s \equiv wu$ with $v \equiv w*^{\varphi}$ and $x \notin \mathrm{FV}(w)$: Then, since w is reducible by hypothesis (2(b)), $s \equiv wu$ is reducible.
6. $\Delta \subseteq v$ and $s \equiv (\lambda x.\, v')u$ with $v \to v'$: Then, by (CR2), v' is reducible. By the WF induction hypothesis, $s \equiv (\lambda x.\, v')u$ is reducible.
7. $\Delta \subseteq u$ and $s \equiv (\lambda x.\, v)u'$ with $u \to u'$: Then, by (CR2), u' is reducible. By the WF induction hypothesis, $s \equiv (\lambda x.\, v)u'$ is reducible.

In every case, the neutral term $(\lambda x.\, v)u$ reduces to reducible terms only. So, by (CR3), $(\lambda x.\, v)u$ is reducible. Hence $\lambda x.\, v$ is reducible. ◀

In the following two theorems, we use Lemma 3. For a term t, a sequence $\vec{x}$ of distinct variables $x_1^{\varphi_1}, \ldots, x_n^{\varphi_n}$, and a sequence $\vec{u}$ of terms $u_1^{\varphi_1}, \ldots, u_n^{\varphi_n}$, let $t[\vec{x} := \vec{u}]$ be the simultaneous substitution.

▶ **Theorem 9** (Restricted Reducibility). *Assume that*

1. *t is a $*$-free term;*
2. *a sequence of distinct variables $x_1^{\varphi_1}, \ldots, x_n^{\varphi_n}$ contains all free variables of t; and*
3. *$u_i^{\varphi_i}$ is reducible and $*$-free $(i = 1, \ldots, n)$.*

Then $t[x_1^{\varphi_1}, \ldots, x_n^{\varphi_n} := u_1^{\varphi_1}, \ldots, u_n^{\varphi_n}]$ is reducible.

Proof. We prove that $t\,[\vec{x} := \vec{u}\,]$ is reducible, by induction on t. As t is $*$-free, $t \not\equiv *^{\top}$. So, we have five cases.

1. $t \equiv x_i$: Then $t\,[\vec{x} := \vec{u}\,] \equiv u_i$. Immediate.
2. $t \equiv \pi_i w$ $(i = 1, 2)$: Then by induction hypothesis, $w\,[\vec{x} := \vec{u}\,]$ is reducible. So each $\pi_i(w\,[\vec{x} := \vec{u}\,])$ is reducible. This term is $\pi_i w\,[\vec{x} := \vec{u}\,] \equiv t\,[\vec{x} := \vec{u}\,]$.
3. $t \equiv \langle u, v\rangle$: Then $t\,[\vec{x} := \vec{u}\,] \equiv \langle u\,[\vec{x} := \vec{u}\,], v\,[\vec{x} := \vec{u}\,]\rangle$. By the induction hypotheses, both $u\,[\vec{x} := \vec{u}\,]$ and $v\,[\vec{x} := \vec{u}\,]$ are $*$-free and reducible. By Lemma 8(1), $t\,[\vec{x} := \vec{u}\,]$, that is, $\langle u\,[\vec{x} := \vec{u}\,], (v\,[\vec{x} := \vec{u}\,])\rangle$, is reducible.
4. $t \equiv wv$: Then by induction hypotheses $w\,[\vec{x} := \vec{u}\,]$ and $v\,[\vec{x} := \vec{u}\,]$ are reducible, and so (by definition) is $w\,[\vec{x} := \vec{u}\,]\,(v\,[\vec{x} := \vec{u}\,])$; but this term is $t\,[\vec{x} := \vec{u}\,]$.
5. $t \equiv \lambda y^{\varphi}.\, w^{\psi}$ with y not free in any $\vec{x}, \vec{u}$: Then $t\,[\vec{x} := \vec{u}\,] \equiv \lambda y.\,(w\,[\vec{x} := \vec{u}\,])$. Let u^{φ} be a reducible, possibly non-$*$-free term. w and $\tilde{u}$ are $*$-free. By induction hypothesis, a $*$-free term $w[\vec{x}, y := \vec{u}, \tilde{u}] \equiv w\,[\vec{x} := \vec{u}\,]\,[y := \tilde{u}]$, is reducible. The last term is $\tilde{v}$ where

$v \equiv w\,[\,\vec{x} := \vec{u}\,]\,[y := u]$, because $w, \vec{u}$ are $*$-free. By Lemma 3, $v \equiv w\,[\,\vec{x} := \vec{u}\,]\,[y := u]$ is reducible. $w\,[\,\vec{x} := \vec{u}\,]$ is $*$-free. So, by Lemma 8(2), $t\,[\,\vec{x} := \vec{u}\,] \equiv \lambda y.\,(w\,[\,\vec{x} := \vec{u}\,])$ is reducible.

Hence we have established the restricted reducibility theorem. ◀

Again we use Lemma 3.

▶ **Theorem 10.** *All terms of* $(\lambda\beta\eta\pi*)'$ *are reducible.*

Proof. Let t be a term. The $*$-free term $\tilde{t}$ is reducible, by (CR4) and by Theorem 9 with $u_i := x_i$, the identity substitution. By Lemma 3, t is reducible. ◀

▶ **Corollary 11.** $(\lambda\beta\eta\pi*)'$ *satisfies SN.*

Proof. By (CR1) and Theorem 10, every term of $(\lambda\beta\eta\pi*)'$ is SN. ◀

▶ Remark. The ordinal number assignment of Howard [18] (Schütte [34], resp.) to typed λ-terms (typed combinators, resp.) proves SN of typed β-reduction (typed combinatory reduction, resp.). Beckmann used cut-elimination procedure [5] of a deduction system to give an optimal upper bound of typed $\beta\eta$-reduction. But these two proofs seem not to generalize for SN of the rewriting system $(\lambda\beta\eta\pi*)'$. In these two proofs, it is not the case that (1) $r*^{\tau} > r$ and (2) the LHS $\lambda x^{\tau}.\,t*^{\tau}$ $(x \notin \mathrm{FV}(t))$ of the rewrite rule schema (η_{top}) is greater than the RHS t.

One may suppose that the higher-order recursive path ordering (HORPO for short) [21] or the General Schema [7], could be extended with surjective pairing and hence be used for proving SN of $(\lambda\beta\eta\pi*)'$. If there is a convenient translation of the rewrite rule schemata $(g), (\eta_{top})$, and (SP_{top}) with type-abstraction to an infinite simply-typed system, such that the translation can also put all the rules of $(\lambda\beta\eta\pi*)'$ in the right kind of format, it is possible that a HORPO-variant (with minimal symbol *) may handle $(\lambda\beta\eta\pi*)'$. However, we need a new HORPO variant, since the conventional ones are troubled with the non-left-linear (SP)-rule `pair(p1(X), p2(X))->X`. There is no type ordering that allows for the extraction of `X` from terms of smaller type in general. The top rule (g): $u^{\tau} \to *^{\tau}$ $(\tau \in \mathit{Iso}(\top),\ u \not\equiv *^{\tau})$ is also problematic for most HORPO-variants. It could be handled by using a variation of HORPO with minimal symbols, such as the one used in WANDA [24]. Here, WANDA is one of the most powerful automatic termination provers for higher-order rewriting.

3 SN proof of polymorphic extensions by restricted reducibility theorem

In [9], Curien and Di Cosmo introduced the polymorphic extension $\lambda^2\beta\eta\pi*$ of the equational theory $\lambda\beta\eta\pi*$, and the polymorphic extension $\left(\lambda^2\beta\eta\pi*\right)'$ of the rewriting system $(\lambda\beta\eta\pi*)'$. We introduce an extension $(\lambda^2\beta\eta\pi*)''$ of $(\lambda\beta\eta\pi*)'$ by *polymorphism* and *terminal types caused by parametric polymorphism* [17].

▶ **Definition 12** ($\left(\lambda^2\beta\eta\pi*\right)'$)**.** We will first recall the equational theory $\lambda^2\beta\eta\pi*$ of the polymorphic terms. The types are generated from type variables $X, Y, \ldots$ and the distinguished type constant $\top$ by means of the product type $\varphi \times \psi$, the function type $\varphi \to \psi$, and the $\Pi X.\,\varphi$. Terms are built up from the distinguished term constant $*^{\top}$ and term variables $x^{\varphi}, y^{\varphi}, \ldots, x^{\psi}, y^{\psi}, \ldots$ by means of λ-abstraction $(\lambda x^{\varphi}.\,t^{\psi})^{\varphi\to\psi}$, term application $(u^{\varphi\to\psi}v^{\varphi})^{\psi}$, pairing $\langle u^{\varphi}, v^{\psi}\rangle^{\varphi\times\psi}$, left-projection $(\pi_1 t^{\varphi\times\psi})^{\varphi}$, right-projection $(\pi_2 t^{\varphi\times\psi})^{\psi}$,

- *universal abstraction*: if v^{φ} is a term, then so is $(\Lambda X.\,v^{\varphi})^{\Pi X.\,\varphi}$, whenever the variable X is not free in the type of a free variable of v^{φ}; and
- *universal application*: if $t^{\Pi X.\,\varphi}$ and ψ is a type, then so is $\left(t^{\Pi X.\,\varphi}\psi\right)^{\varphi[X:=\psi]}$.

The superscript representing the type is often omitted. The axioms of the equational theory $\lambda^2\beta\eta\pi*$ are those of $\lambda\beta\eta\pi*$ and the following two:

$$(\beta^2)\ \ (\Lambda X.\, t)\varphi = t[X := \varphi]. \qquad (\eta^2)\ \ \Lambda X.\, tX = t,\ \ (X \text{ does not occur free in } t).$$

For the definition of $Iso(\top)$ for $\left(\lambda^2\beta\eta\pi*\right)'$, Curien and Di Cosmo added the following clause to the inductive definition of $Iso(\top)$ for $(\lambda\beta\eta\pi*)'$:
($\top\mathbf{\Pi}$) If τ is "isomorphic to" $\top$, so is $\Pi X.\, \tau$. The canonical term $*^{\Pi X.\, \tau}$ of $\Pi X.\, \tau$ is $\Lambda X.\, *^{\tau}$.

The rewrite rule schemata of the rewriting system $\left(\lambda^2\beta\eta\pi*\right)'$ are those of the rewriting system $(\lambda\beta\eta\pi*)'$ and those obtained from (β^2) and (η^2) by orienting left to right. This completes the definition of $\left(\lambda^2\beta\eta\pi*\right)'$.

Taking the parametricity of the polymorphism [17] into account, we add the following clause to the inductive definition of $Iso(\top)$:
($\top^{par}$) For every $n \geq 0$, if $\tau_1, \ldots, \tau_n$ are "isomorphic to" $\top$, so is $\Pi X.\, ((\tau_1 \to \cdots \to \tau_n \to X) \to X)$. The canonical term $*^{\Pi X.\, ((\tau_1 \to \cdots \to \tau_n \to X) \to X)}$ of $\Pi X.\, ((\tau_1 \to \cdots \to \tau_n \to X) \to X)$ is $\Lambda X. \lambda x^{\tau_1 \to \cdots \tau_n \to X}.\, x\, *^{\tau_1} \cdots *^{\tau_n}$.
As (g)-rule schema applies for more terms in $\left(\lambda^2\beta\eta\pi*\right)''$ than in $\left(\lambda^2\beta\eta\pi*\right)'$, SN of $\left(\lambda^2\beta\eta\pi*\right)''$ implies SN of $\left(\lambda^2\beta\eta\pi*\right)'$. We will prove SN of $\left(\lambda^2\beta\eta\pi*\right)''$.

In [9], to show SN of the rewriting system $\left(\lambda^2\beta\eta\pi*\right)'$, Curien and Di Cosmo tried to prove that every term of $\left(\lambda^2\beta\eta\pi*\right)'$ in the g-normal form is SN. But they observed that the set of g-normal form is not closed under β^2-reduction; $(\Lambda X.\, \lambda x^X.\, \lambda y^{X\to Y}.\, yx)\top$ is in g-normal form, but its reduct $u \equiv \lambda x^{\top}.\, \lambda y^{\top\to Y}.\, yx$ is not, as $u \to_g \lambda x^{\top}.\, \lambda y^{\top\to Y}.\, y*$.

▶ **Definition 13** (Neutral). A term is *neutral* if it is not of the form $\langle u, v\rangle$, $\lambda x.\, v$, or $\Lambda X.\, u$.

As in $(\lambda\beta\eta\pi*)'$, we consider (CR0) to define a *reducibility candidate* [12].

▶ **Definition 14.** A *reducibility candidate* (RC for short) of type φ is a set $\mathcal{R}$ of terms of type φ such that:
(CR0) If $\varphi \in Iso(\top)$, then $*^{\varphi} \in \mathcal{R}$.
(CR1) If $t^{\varphi} \in \mathcal{R}$, then t^{φ} is SN.
(CR2) If $t^{\varphi} \in \mathcal{R}$ and $t^{\varphi} \to t'$, then $t' \in \mathcal{R}$.
(CR3) If t^{φ} is neutral, and any reduct of t^{φ} is in $\mathcal{R}$, then $t^{\varphi} \in \mathcal{R}$.

▶ **Lemma 15.** *(CR0) and (CR3) implies*
(CR4) *If t^{φ} is a variable, then t is in $\mathcal{R}$.*

▶ **Definition 16.**
1. Let $\mathcal{SN}^{\psi}$ be the set of SN terms of type ψ.
2. For an RC $\mathcal{R}$ of type φ and an RC $\mathcal{S}$ of type ψ, define

$$\mathcal{R} \times \mathcal{S} = \{t^{\varphi\times\psi} \mid \pi_1 t \in \mathcal{R},\ \pi_2 t \in \mathcal{S}\},\ \text{and}\ \mathcal{R} \to \mathcal{S} = \{t^{\varphi\to\psi} \mid \forall u(u \in \mathcal{R} \implies tu \in \mathcal{S})\}.$$

▶ **Lemma 17.**
1. *For any type ψ, $\mathcal{SN}^{\psi}$ is an RC.*
2. *If $\mathcal{R}, \mathcal{S}$ are RCs of type φ, ψ, then $\mathcal{R} \times \mathcal{S}, \mathcal{R} \to \mathcal{S}$ are RCs of type $\varphi \times \psi, \varphi \to \psi$.*

Proof. (1) (CR0): $*^{\psi} \in \mathcal{SN}^{\psi}$ is SN, if $\psi \in Iso(\top)$. (CR1): By the definition of $\mathcal{SN}^{\psi}$. (CR2): If $t \in \mathcal{SN}^{\psi}$ and $t \to t'$, then $t' \in \mathcal{SN}^{\psi}$. (CR3): Let t be a neutral term of type ψ such that any reduct t' of t is in $\mathcal{SN}^{\psi}$. Then t is in $\mathcal{SN}^{\psi}$. (2) By the proof of Lemma 6 for $\varphi \times \psi$ and $\varphi \to \psi$. ◀

For a type φ, a sequence $\vec{X}$ of distinct type variables $X_1, \ldots, X_m$, and a sequence $\vec{\psi}$ of types $\psi_1, \ldots, \psi_m$, let $\varphi[\vec{X} := \vec{\psi}]$ be the simultaneous substitution.

▶ **Definition 18** (parametric reducibility)**.** Suppose

1. φ is a type;
2. a sequence $\vec{X}$ of distinct type variables $X_1, \ldots, X_m$ contains all free type variables of φ;
3. $\vec{\psi}$ is a sequence of types $\psi_1, \ldots, \psi_m$; and
4. $\vec{\mathcal{R}}$ is a sequence of RCs $\mathcal{R}_1, \ldots, \mathcal{R}_m$ of corresponding types $\vec{\psi}$.

Define a set $\mathsf{RED}_\varphi[\vec{X} := \vec{\mathcal{R}}]$ of terms of type $\varphi[\vec{X} := \vec{\psi}]$ as follows:

1. If $\varphi = \top$, $\mathsf{RED}_\varphi[\vec{X} := \vec{\mathcal{R}}] = \mathcal{SN}^\top$;
2. If $\varphi = X_i$, $\mathsf{RED}_\varphi[\vec{X} := \vec{\mathcal{R}}] = \mathcal{R}_i$;
3. If $\varphi \equiv \varphi' \# \varphi''$, $\mathsf{RED}_\varphi[\vec{X} := \vec{\mathcal{R}}] = \mathsf{RED}_{\varphi'}[\vec{X} := \vec{\mathcal{R}}] \mathbin{\#} \mathsf{RED}_{\varphi''}[\vec{X} := \vec{\mathcal{R}}]$ $(\# = \to, \times)$;
4. If $\varphi \equiv \Pi Y.\, \varphi'$, Y not free in $\vec{\psi}$ and $Y \neq X_i$ $(i = 1, \ldots, m)$, then $\mathsf{RED}_\varphi[\vec{X} := \vec{\mathcal{R}}]$ is the set of terms $t^{\Pi Y.\, \varphi'[\vec{X} := \vec{\psi}]}$ such that for any type ψ and any RC $\mathcal{S}$ of type ψ, $(t\psi)^{\varphi'[\vec{X}, Y := \vec{\psi}, \psi]} \in \mathsf{RED}_{\varphi'}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}]$.

▶ **Lemma 19.** *Under the conditions of Definition 18,* $\mathsf{RED}_\varphi[\vec{X} := \vec{\mathcal{R}}]$ *is an RC of type* $\varphi[\vec{X} := \vec{\psi}]$.

Proof. By induction on φ. First consider the case $\varphi \equiv \Pi Y.\, \varphi'$. Let $\mathcal{S}$ be an RC $\mathcal{S}$ of type φ''. By induction hypothesis,

$$\mathcal{T} := \vec{\mathsf{RED}}_{\varphi'}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}], \text{ is an RC.} \tag{3}$$

(CR0) Let $*^{\Pi Y.\, \varphi'[\vec{X} := \vec{\psi}]}\varphi'' \to s$. We will verify $s \in \mathcal{T}$. We have two cases. The first case corresponds to clause $(\top\Pi)$ and the second case to clause $(\top^{par})$ in Definition 12.

1. $s \equiv *^{\varphi'[\vec{X}, Y := \vec{\psi}, \varphi'']}$. By (3), $s \in \mathcal{T}$.
2. $s \equiv \lambda y.\, y\, *^{\tau_1} \cdots *^{\tau_n}$: Then $\varphi' = (\tau_1 \to \cdots \to \tau_n \to Y) \to Y$. We will verify $s \in \mathsf{RED}_{\varphi'}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}] = (\mathsf{RED}_{\tau_1}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}] \to \cdots \to \mathsf{RED}_{\tau_n}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}] \to \mathcal{S}) \to \mathcal{S}$. Take a term

$$u^{\tau_1 \to \cdots \to \tau_n \to \varphi''} \in \mathsf{RED}_{\tau_1}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}] \to \cdots \to \mathsf{RED}_{\tau_n}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}] \to \mathcal{S}. \tag{4}$$

By induction hypothesis for τ_i,

$$\mathsf{RED}_{\tau_i}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}] \text{ is an RC.} \tag{5}$$

By Lemma 17 (2), $\mathsf{RED}_{\tau_1}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}] \to \cdots \to \mathsf{RED}_{\tau_n}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}] \to \mathcal{S}$ is an RC. By (CR1) of this RC, u is SN. So we can use WFI $\left(\left(\left\{u^{\tau_1 \to \cdots \to \tau_n \to \varphi''} \mid (4) \text{ holds}\right\}, \to\right)\right)$ where $\to$ is the rewrite relation. We will prove $su \equiv (\lambda y.\, y\, *^{\tau_1} \cdots *^{\tau_n})u \in \mathcal{S}$. Let $(su)^{\varphi''} \xrightarrow{\Delta} v$. Then we have four subcases:

   a. $\Delta \equiv su$ is a redex of (g): Then v is $*^{\varphi''}$. As $\mathcal{S}$ is an RC, $v \in \mathcal{S}$ by (CR0) of $\mathcal{S}$.
   b. $\Delta \equiv su$ is not a redex of (g): Then $v \equiv u\, *^{\tau_1} \cdots *^{\tau_n}$. By (5) and the (CR0), $*^{\tau_i} \in \mathsf{RED}_{\tau_i}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}]$. So $v \in \mathcal{S}$ by (4).
   c. $\Delta \subseteq s$: Then $v \equiv \lambda y.\, *^\tau *^{\tau_{n-i+1}} \cdots *^{\tau_n}$ for some τ and nonnegative $i \leq n$ with the rewrite rule schema is (g). We see that the normal form of s is canonical. This canonical term is in $\mathcal{T}$ by (3) and (CR0) of $\mathcal{T}$. By repeated applications of induction hypothesis (CR3), $s \in \mathcal{T}$. Hence $v \in \mathcal{S}$.
   d. Otherwise, $\Delta \subseteq u$. By the WF induction hypothesis, $v \in \mathcal{S}$.

   So, $v \in \mathcal{S}$. By (CR3) for $\mathcal{S}$, $(\lambda y.\, y\, *^{\tau_1} \cdots *^{\tau_n})u \in \mathcal{S}$. So, $s \in \mathcal{T}$.

Thus $*^{\Pi Y.\,\varphi'[\vec{X}:=\vec{\psi}]}\varphi'' \in \mathcal{T}$. Hence $*^{\Pi Y.\,\varphi'[\vec{X}:=\vec{\psi}]} \in \mathsf{RED}_{\Pi Y.\,\varphi'}[\vec{X} := \vec{\mathcal{R}}]$.

(CR1) Let $t \in \mathsf{RED}_{\varphi}[\vec{X} := \vec{\mathcal{R}}]$. Then $t\varphi'' \in \mathcal{T}$ by Definition 18. By (3) and (CR1) of $\mathcal{T}$, $t\varphi''$ is SN. So t is SN.

(CR2) Let $t \in \mathsf{RED}_{\varphi}[\vec{X} := \vec{\mathcal{R}}]$. Then $t\varphi'' \in \mathcal{T}$ by Definition 18. Assume $t \to t'$. Then $t\varphi'' \to t'\varphi''$. By (3) and (CR2) of $\mathcal{T}$, $t'\varphi'' \in \mathcal{T}$. So $t' \in \mathsf{RED}_{\varphi}[\vec{X} := \vec{\mathcal{R}}]$.

(CR3) Suppose that t is neutral and that $t' \in \mathsf{RED}_{\varphi}[\vec{X} := \vec{\mathcal{R}}]$ whenever $t \to t'$. Let $t\varphi'' \xrightarrow{\Delta} s$. As t is neutral, we have two cases:

1. $\Delta \equiv t\varphi''$: Then s is canonical of type $\varphi'[\vec{X} := \vec{\psi}]$, because t is neutral. By (3) and (CR0) of $\mathcal{T}$, $s \in \mathcal{T}$.
2. Otherwise, $s \equiv t'\varphi''$ with $t \xrightarrow{\Delta} t'$. $s \in \mathcal{T}$ by $t' \in \mathsf{RED}_{\varphi}[\vec{X} := \vec{\mathcal{R}}]$.

By (3) and (CR3) of $\mathcal{T}$, $t\varphi'' \in \mathcal{T}$. So $t \in \mathsf{RED}_{\varphi}[\vec{X} := \vec{\mathcal{R}}]$.

The cases where φ is other than $\Pi Y.\,\varphi'$ are by induction hypotheses on φ and Lemma 17. ◀

▶ **Lemma 20.** *Suppose that*

1. *φ, ψ are types, Y is a type variable;*
2. *a sequence $\vec{X}$ of distinct type variables $X_1, \ldots, X_m$ contains all free type variables of $\varphi[Y := \psi]$;*
3. *$X_i \neq Y$ $(i = 1, \ldots, m)$; and*
4. *$\vec{\mathcal{R}}$ is a sequence of RCs $\mathcal{R}_1, \ldots, \mathcal{R}_m$.*

Then

$$\mathsf{RED}_{\varphi[Y:=\psi]}[\vec{X} := \vec{\mathcal{R}}] = \mathsf{RED}_{\varphi}[\vec{X}, Y := \vec{\mathcal{R}}, \mathsf{RED}_{\psi}[\vec{X} := \vec{\mathcal{R}}]].$$

Proof. By induction on φ. ◀

▶ **Lemma 21** (Universal abstraction)**.** *Suppose that*

1. *φ is a type;*
2. *a sequence $\vec{X}$ of distinct type variables $X_1, \ldots, X_m$ contains all free type variables of $\Pi Y.\,\varphi$;*
3. *$X_i \neq Y$ $(i = 1, \ldots, m)$, $\vec{\psi}$ is a sequence of types $\psi_1, \ldots, \psi_m$;*
4. *$\vec{\mathcal{R}}$ is a sequence of RCs $\mathcal{R}_1, \ldots, \mathcal{R}_m$ of types $\vec{\psi}$;*
5. *Y does not occur free in $\vec{\psi}$; and*
6. *$w^{\varphi[\vec{X}:=\vec{\psi}]}$ is a term.*

If for any type ψ and any RC $\mathcal{S}$ of type ψ, $(w[Y := \psi])^{\varphi[\vec{X},Y:=\vec{\psi},\psi]} \in \mathsf{RED}_{\varphi}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}]$, then $\Lambda Y.\,w \in \mathsf{RED}_{\Pi Y.\,\varphi}[\vec{X} := \vec{\mathcal{R}}]$.

Proof. $\mathcal{SN}^Y$ is an RC, by Lemma 17 (1). By assumption, $w \in \mathsf{RED}_{\varphi}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{SN}^Y]$. By (CR1) of this RC, w is SN. By Definition 18 (4), we have only to verify:

$$(\Lambda Y.\,w)\psi \in \mathsf{RED}_{\varphi}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}], \text{for every type } \psi \text{ and RC } \mathcal{S} \text{ of type } \psi. \tag{6}$$

The proof is by WFI $\left(\left(\left\{w^{\varphi[\vec{X}:=\vec{\psi}]} \mid\ w \in \mathsf{RED}_{\varphi}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{SN}^Y]\right\},\ \to\right)\right)$ where $\to$ is the rewrite relation. Let $(\Lambda Y.\,w)\psi \xrightarrow{\Delta} s$. We have five cases. We verify $s \in \mathcal{T} := \mathsf{RED}_{\varphi}[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}]$.

1. $\Delta \equiv (\Lambda Y.\,w)\psi$ is a redex of (g): Then $s \equiv *^{\varphi[\vec{X},Y:=\vec{\psi},\psi]}$. By (CR0) of $\mathcal{T}$.
2. $\Delta \equiv (\Lambda Y.\,w)\psi$ is a redex of (β^2): Then $s \equiv w[Y := \psi]$. By assumption.
3. $\Delta \equiv (\Lambda Y.\,w)$ is a redex of (g): Then $s \equiv *^{\Pi Y.\,\varphi[\vec{X}:=\vec{\psi}]}\psi$. By $*^{\Pi Y.\,\varphi[\vec{X}:=\vec{\psi}]} \in \mathsf{RED}_{\Pi Y.\,\varphi}[\vec{X} := \vec{\mathcal{R}}]$, $s \in \mathcal{T}$ by Definition 18.
4. $\Delta \equiv (\Lambda Y.\,w)$ is a redex of (η^2): Then this case coincides with the second case.

5. Otherwise, for some w', $s \equiv (\Lambda Y.\, w')\psi$ and $w \to w'$. By the WF induction hypothesis. Thus $s \in \mathcal{T}$. So (6) follows from (CR3) of $\mathcal{T}$. ◀

▶ **Lemma 22** (Universal application)**.** *Suppose that*
1. *φ, ψ are types, Y is a type variable;*
2. *a sequence $\vec{X}$ of distinct type variables $X_1, \ldots, X_m$ contains all free type variables of $\varphi[Y := \psi]$;*
3. *$X_i \neq Y$ $(i = 1, \ldots, m)$, $\vec{\psi}$ is a sequence of types $\psi_1, \ldots, \psi_m$; and*
4. *$\vec{\mathcal{R}}$ is a sequence of RCs $\mathcal{R}_1, \ldots, \mathcal{R}_m$ of types $\vec{\psi}$.*

Then[1]

$$w \in \mathsf{RED}_{\Pi Y.\,\varphi}[\vec{X} := \vec{\mathcal{R}}] \implies w\left(\psi[\vec{X} := \vec{\psi}]\right) \in \mathsf{RED}_{\varphi[Y:=\psi]}[\vec{X} := \vec{\mathcal{R}}].$$

Proof. By Lemma 19, $\mathsf{RED}_{\psi}[\vec{X} := \vec{\mathcal{R}}]$ is an RC of type $\psi[\vec{X} := \vec{\psi}]$. By the premise and Definition 18 (4), $w\left(\psi[\vec{X} := \vec{\psi}]\right) \in \mathsf{RED}_{\varphi}[\vec{X}, Y := \vec{\mathcal{R}}, \mathsf{RED}_{\psi}[\vec{X} := \vec{\mathcal{R}}]\,]$. So Lemma 20 implies the conclusion. ◀

By the condition $(\top^{par})$, a canonical term $*^{\tau}$ $(\tau \in \mathit{Iso}(\top))$ does not necessarily contain the term constant $*^{\top}$. We generalize the definition of $*$-free and $\tilde{t}$ (Definition 2.)

▶ **Definition 23.**
1. We say a term of $\left(\lambda^2\beta\eta\pi*\right)''$ is *$*$-free* , if it has no canonical subterm of a type of $\mathit{Iso}(\top)$.
2. For any term t of $\left(\lambda^2\beta\eta\pi*\right)''$, let $\tilde{t}$ be a $*$-free term obtained from t by replacing all occurrences of $*^{\tau}$ with variables x^{τ} where τ is any type of $\mathit{Iso}(\top)$.

Similarly as Lemma 3, we can prove:

▶ **Lemma 24.** *Let φ be a type, $\mathcal{R}$ be an RC of φ, and t be a term of $\left(\lambda^2\beta\eta\pi*\right)''$ of φ. If $\tilde{t} \in \mathcal{R}$, $t \in \mathcal{R}$.*

▶ **Lemma 25.** *In $\left(\lambda^2\beta\eta\pi*\right)''$, for every $*$-free term t,*
1. *$t[X_1, \ldots, X_m := \psi_1, \ldots, \psi_m]$ is $*$-free for all distinct type variables $X_1, \ldots, X_m$ and for all types $\psi_1, \ldots, \psi_m$; and*
2. *$t[x_1^{\varphi_1}, \ldots, x_n^{\varphi_n} := u_1^{\varphi_1}, \ldots, u_n^{\varphi_n}]$ is $*$-free for all distinct variables $x_1^{\varphi_1}, \ldots, x_n^{\varphi_n}$ and for all $*$-free terms $u_1^{\varphi_1}, \ldots, u_n^{\varphi_n}$.*

Proof. By induction on t. Let Θ be $[X_1, \ldots, X_m := \psi_1, \ldots, \psi_m]$ and θ be $[x_1^{\varphi_1}, \ldots, x_n^{\varphi_n} := u_1^{\varphi_1}, \ldots, u_n^{\varphi_n}]$. The proof proceeds by cases according to the form of t. By Definition 12, t is not a term constant, because otherwise t is $*^{\top}$.
1. t is a variable: Then (1) holds because no canonical term has a free variable. (2) is clear.
2. t is an abstraction, or an application: Obvious.
3. $t \equiv \Lambda Y.\, w$ such that $X_i \not\equiv Y$ and Y does not occur free in any ψ_i: By induction hypothesis, $w\Theta$ and $w\theta$ are $*$-free. (1) We have only to verify that $t\Theta \equiv \Lambda Y.\, w\Theta$ is not a canonical term:

$$\Lambda Y.\lambda x^{\tau_1 \to \cdots \tau_l \to Y}.\, x\, *^{\tau_1} \cdots *^{\tau_l} \text{ where } \tau_k \in \mathit{Iso}(\top)\ (k = 1, \ldots, l). \tag{7}$$

Let τ_i be an instance of some type σ by the substitution Θ such that $\sigma \not\equiv \tau$. Then $\sigma \notin \mathit{Iso}(\top)$ because any type of $\mathit{Iso}(\top)$ has no free type variable. So, $*^{\tau_i}$ is an instance

[1] [12, Lemma 14.2.3] corresponding to this lemma has a typo: "tV" should be "$t(V[\underline{U}/\underline{X}])$."

of a non-canonical subterm of t by the substitution Θ. This contradicts against the induction hypothesis. Hence, $t\Theta$ is $\equiv t$, which is $*$-free by the premise.

(2) Assume $t\theta \equiv \Lambda Y.\, w\theta$ is (7). By the premise, no u_j contains $*^{\tau_k}$. We have two subcases.

a. $l = 0$: Then $t\theta \equiv t$ or $t \equiv \Lambda Y.\, x_j^{Y \to Y}$. In the former case, $t\theta$ is $*$-free. The latter case is impossible because of the proviso of *universal abstraction* in Definition 12.

b. $l > 0$: Then, for each k, there is a $*$-free subterm w_k of t such that $w_k\theta \equiv *^{\tau_k}$. This is impossible by induction hypothesis for w_k.

Hence $t\theta$ is $*$-free.

4. $t \equiv w\psi$: Then, by induction hypothesis, $w\Theta$ and $w\theta$ are $*$-free. Hence, none of $t\Theta \equiv w\Theta(\psi\Theta)$ and $t\theta \equiv (w\theta)\psi$ is canonical. ◀

In the following two theorems, we use Lemma 24.

▶ **Theorem 26** (Restricted Reducibility)**.** *Suppose*

1. t^φ *is a* $*$-free *term;*
2. *a sequence of distinct variables* $x_1^{\varphi_1}, \dots, x_n^{\varphi_n}$ *contains all free variables of* t^φ*;*
3. *a sequence* $\vec{X}$ *of distinct type variables* $X_1, \dots, X_m$ *contains all free type variables of types* $\varphi, \varphi_1, \dots, \varphi_n$*;*
4. $\vec{\mathcal{R}}$ *is a sequence of RCs* $\mathcal{R}_1, \dots, \mathcal{R}_m$ *of types* $\vec{\psi} \equiv \psi_1, \dots, \psi_m$*; and*
5. $u_i^{\varphi_i[\vec{X}:=\vec{\psi}]}$ *is in* $\mathsf{RED}_{\varphi_i}[\vec{X} := \vec{\mathcal{R}}]$ *and is* $*$-free *(*$i = 1, \dots, n$*).*
6. $t[\vec{X} := \vec{\psi}][\vec{x} := \vec{u}]$ *is the term obtained from* $t[\vec{X} := \vec{\psi}]$ *by simultaneously substitution of* $u_1^{\varphi_1[\vec{X}:=\vec{\psi}]}, \dots, u_n^{\varphi_n[\vec{X}:=\vec{\psi}]}$ *into* $x_1^{\varphi_1[\vec{X}:=\vec{\psi}]}, \dots, x_n^{\varphi_n[\vec{X}:=\vec{\psi}]}$.

Then $t[\vec{X} := \vec{\psi}][\vec{x} := \vec{u}]$ *is in* $\mathsf{RED}_\varphi[\vec{X} := \vec{\mathcal{R}}]$.

Proof. By induction on t. The proof proceeds by cases according to the form of t.

1. t is a pairing or a λ-abstraction: We can prove this case, similarly as in the proof of Theorem 9, but we use Lemma 25 to verify the $*$-free condition of Lemma 8.
2. $t \equiv (\Lambda Y.\, w)^{\Pi Y.\, \varphi}$ where $X_i \neq Y$ and Y does not occur free in any $\varphi_i[\vec{X} := \vec{\psi}]$: Then by the induction hypothesis, for any type ψ and any RC $\mathcal{S}$ of ψ, $w[\vec{X}, Y := \vec{\psi}, \psi][\vec{x} := \vec{u}]$ is in $\mathsf{RED}_\varphi[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}]$. Since Y occurs in no $\vec{u}$, we have $w[\vec{X} := \vec{\psi}][\vec{x} := \vec{u}][Y := \psi] \in \mathsf{RED}_\varphi[\vec{X}, Y := \vec{\mathcal{R}}, \mathcal{S}]$. By Lemma 21, $(\Lambda Y.\, w)[\vec{X} := \vec{\psi}][\vec{x} := \vec{u}]$ is in $\mathsf{RED}_{\Pi Y.\, \varphi}[\vec{X} := \vec{\mathcal{R}}]$.
3. $t \equiv w^{\Pi Y.\, \varphi}\psi$: Then by the induction hypothesis, $w[\vec{X} := \vec{\psi}][\vec{x} := \vec{u}]$ is in $\mathsf{RED}_{\Pi Y.\, \varphi}[\vec{X} := \vec{\mathcal{R}}]$. By Lemma 22, $w[\vec{X} := \vec{\psi}][\vec{x} := \vec{u}]\left(\psi[\vec{X} := \vec{\psi}]\right) \in \mathsf{RED}_{\varphi[Y:=\psi]}[\vec{X} := \vec{\mathcal{R}}]$. This term is just $(w\psi)[\vec{X} := \vec{\psi}][\vec{x} := \vec{u}]$.

The other cases are handled similarly as in the proof of Theorem 9, except that we use Lemma 24 instead of Lemma 3. ◀

▶ **Theorem 27.** *If a sequence of distinct type variables* $X_1, \dots, X_m$ *contains the free type variables of a type* φ*, then any term* t^φ *is in* $\mathsf{RED}_\varphi[X_1, \dots, X_m := \mathcal{SN}^{X_1}, \dots, \mathcal{SN}^{X_m}]$.

Proof. Let t be a term. The $*$-free term $\tilde{t}$ is reducible, by (CR4) and by Theorem 26 with $u_i^{\varphi_i} := x_i^{\varphi_i}$, $\psi_j := X_j$ and $\mathcal{R}_j := \mathcal{SN}^{X_j}$. By Lemma 24, t is reducible. ◀

▶ **Corollary 28.** *(1)* $(\lambda^2\beta\eta\pi*)''$ *is SN. (2)* $\left(\lambda^2\beta\eta\pi*\right)'$ *satisfies SN.*

Proof. (1) By (CR1) and Theorem 27. (2) If $u \to v$ in $\left(\lambda^2\beta\eta\pi*\right)'$, then $u \to v$ in $\left(\lambda^2\beta\eta\pi*\right)''$. So $\left(\lambda^2\beta\eta\pi*\right)'$ satisfies SN by (1). ◀

4 Derivational complexity of $(\lambda\beta\eta\pi*)'$: comparison to type-directed expansions

For the typed λ-calculus, let a binary relation $\to_{\overline{\eta}}$ ($\to_{\overline{SP}}$) replace a neutral subterm occurrence in a non-elimination context with the η (SP)-expansion [29]. Neither $\to_{\overline{\eta}}$ nor $\to_{\overline{SP}}$ is stable under contexts. We call the relation $\to := \to_{\beta\pi_1\pi_2 T\overline{\eta}\overline{SP}}$ *Mints' reduction*, as Mints introduced it in [28, 29]. Mints' reduction generates the equational theory $\lambda\beta\eta\pi*$, and is SN+CR ([3, 20] to cite a few). In [3], the author presented a divide-and-conquer lemma to infer SN+CR property of a reduction system from that property of its subsystems. From this lemma, SN+CR of the $\to$ follows. SN implies $\to_{\overline{\eta}} = \leftarrow_{\eta} \setminus \leftarrow_{\beta}$ and $\to_{\overline{SP}} = \leftarrow_{SP} \setminus \leftarrow_{\pi_1} \setminus \leftarrow_{\pi_2}$. The divide-and-conquer lemma suggests a large upper bound of the length of a $\to$-reduction sequence. Čubrić proved CR of $\to$ by development argument of residuals [8] and WN. Although $\to_{\overline{\eta}\overline{SP}}$ is not stable under contexts, Khasidashvili and van Oostrom [22] pointed out that the finite development-like argument based on $\leftarrow_{\eta SP}$ proves CR of $\to$.

In [10], Di Cosmo and Kesner proved CR+SN of a reduction system $\to_{\beta} \cup \to_{\overline{\eta}} \cup \to_{\pi_1} \cup \to_{\pi_2} \cup \to_{\overline{SP}} \cup \to_{T}$ union the β-like reductions of sum types. By showing how substitution and the reduction interact with the context-sensitive rules, they proved the WCR. They simulated expansions without expansions, to reduce SN of the reduction to SN for the underlying calculus without expansions, provable by the standard reducibility method.

The rewriting system $(\lambda\beta\eta\pi*)'$ of Curien and Di Cosmo is stable under contexts (i.e., $t \to t' \implies \cdots t \cdots \to \cdots t' \cdots$.) Mints' reduction decides the equational theory $\lambda\beta\eta\pi*$. Mints' reduction is not stable under contexts.

Mints' reduction fits with semantic treatments such as *normalization by evaluation* (e.g., [4]. See [1] in the context of type-checking of dependent type theories). However, because of the complication of Mints' reduction, in his book [30] on selected papers of proof theory, Mints replaced his reduction with the $\beta\eta$-reduction modulo equivalence relation on terms. His purpose is to give a simple proof of difficult theorems of category theory with typed λ-calculus and proof theory by using the correspondence objects = types = propositions and arrows = terms = proofs. Mac Lane is interested in his ambition [27].

In the worst case analysis, normalizing rewriting sequences of Curien and Di Cosmo's rewriting have smaller number of $\beta\pi_1\pi_2 T$-reduction steps than those of Mints' reduction.

The derivational complexity of Mints' reduction is higher than that of Curien-Di Cosmo's rewriting in the simply-typed regime. By the *derivational complexity* of a term t, we mean the maximum number of β-reduction steps in a reduction sequence from t. We count only β-reduction steps, because the β-rule is a common rule of Mints' reduction and the $\beta\eta$-calculus, which is Curien-Di Cosmo's rewriting in the simply-typed regime. The optimal bound for the length of a $\beta\eta$-reduction sequence is given in [5] and is also the optimal bound for the length of a β-reduction sequence.

▶ **Theorem 29.** *For every simply-typed λ-term, the derivational complexity of t of Mints' reduction $\to_{\beta\overline{\eta}}$ is greater than or equal to that of the $\beta\eta$-reduction.*

Proof. A $\beta\eta$-reduction sequence S is an alternating sequence of $\overset{+}{\to}_{\beta}$ and $(\to_{\eta} \setminus \to_{\beta})^{+}$. Here $(\cdots)^{+}$ stands for the transitive closure. By [3], (1) $\to_{\overline{\eta}} = \leftarrow_{\eta} \setminus \leftarrow_{\beta}$; (2) $\to_{\overline{\eta}}$ is CR; and (3) If $t \to_{\beta} s$ then the $\overline{\eta}$-normal form of t goes to that of s in positive number of β-reduction steps. Hence, the $\overline{\eta}$-normal forms of terms in S forms an $\overline{\eta}$-normalization sequence followed by a β-reduction sequence such that the number of β-steps is not less than that of S. ◀

Consider the minimum length ℓ_τ of the normalization sequences of variable x^τ with $\tau \in Iso(\top)$. $\ell_\tau = 1$, in the rewriting system $(\lambda\beta\eta\pi*)'$, although ℓ_τ is arbitrary large as

$\tau \in Iso(\top)$ gets complex in the Mints' reduction. Indeed, $x^\top \to *^\top$,

$$x^{\top\to\top} \to \lambda z^\top.(xz)^\top \to \lambda z^\top.*^\top,$$
$$x^{\top\times\top} \to \langle(\pi_1 x)^\top, (\pi_2 x)^\top\rangle \to \langle(\pi_1 x)^\top, *^\top\rangle \to \langle *^\top, *^\top\rangle.$$

We may be able to introduce SN+CR extensional λ-calculus with surjective pairing, terminal type and empty type, based on type isomorphism.

Lemma 3 and Lemma 24 are described in topological jargon. For any type φ, the reducibility predicate is an open condition, in the topological space of terms where a set of terms is open if and only if the set is closed under reduction. For the set U of $*$-free terms, the maximum open superset of U is the set V of all terms. But U is not dense in V.

Acknowledgements. The author thanks K. Fujita, H. Goguen, D. Kesner, K. Kikuchi, C. Kop, K. Kusakari, F. Pfenning, Y. Toyama, H. Yokouchi, and anonymous referees. The author owes C. Kop for the observation on HORPO and WANDA.

References

1 A. Abel, T. Coquand, and P. Dybjer. Verifying a semantic $\beta\eta$-conversion test for Martin-Löf type theory. In P. Audebaud and C. Paulin-Mohring, editors, *Proceedings of the 9th International Conference on Mathematics of Program Construction (MPC'08)*, pages 29–56, 2008.

2 A. Abel, T. Coquand, and M. Pagano. A modular type-checking algorithm for type theory with singleton types and proof irrelevance. *Logical Meth. in Comput. Sci.*, 7(2:4):1–57, 2011.

3 Y. Akama. On Mints' reductions for ccc-calculus. In M. Bezem and J. Groote, editors, *Proceedings of the 1st International Conference on Typed Lambda Calculus and Applications*, volume 664 of *LNCS*, pages 1–12. Springer-Verlag, 1993.

4 V. Balat, R. Di Cosmo, and M. Fiore. Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. In *Proceedings of the 31st Symposium on Principles of Programming Languages (POPL 2004)*, pages 64–76. ACM Press, Jann 2004.

5 A. Beckmann. Exact bounds for lengths of reductions in typed λ-calculus. *J. Symb. Logic*, 66(3):1277–1285, 2001.

6 F. Blanqui. Computability closure: ten years later. In H. Common-Lundh, C. Kirchner, and H. Kirchner, editors, *Rewriting, Computation and Proof*, volume 4600 of *LNCS*, pages 68–88. Springer, 2007.

7 F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive-data-type systems. *Theoret. Comput. Sci.*, 272:41–68, 2002.

8 D. Čubrić. On free ccc. Distributed on the types mailing list, 1992.

9 P.-L. Curien and R. Di Cosmo. A confluent reduction system for the lambda-calculus with surjective pairing and terminal object. *J. Funct. Programming*, 6(2):299–327, 1996. A preliminary version appeared with the same authors and the same title in: J. Leach Albert, B. Monien, and M. Rodriguez Artalejo, editors, *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, Vol. 510 of LNCS, pages 291–302. Springer-Verlag, 1991.

10 R. Di Cosmo and D. Kesner. Simulating expansions without expansions. *Math. Structures Comput. Sci.*, 4:1–48, 1994.

11 D. Dougherty and R. Subrahmanyam. Equality between functionals in the presence of coproducts. In *Proceedings of the 10th Symposium on Logic in Computer Science*, pages 282–291. IEEE Computer Society Press, 1995.

12 J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1989.

13 H. Goguen. *A Typed Operational Semantics for Type Theory*. PhD thesis, University of Edinburgh, August 1994. Available as LFCS Report ECS-LFCS-94-304.

14 H. Goguen. A syntactic approach to eta equality in type theory. In *Proceedings of the 32nd Symposium on Principles of Programming Languages (POPL 2005)*, pages 75–84. ACM Press, Jan. 2005.

15 M. Haigya. From programming-by-example to proving-by-example. In T. Ito and A. R. Meyer, editors, *Proceedings of Theoretical Aspects of Computer Software*, volume 526 of *LNCS*, pages 387–419. Springer-Verlag, 1991.

16 R. Harper and F. Pfenning. On equivalence and canonical forms in the LF type theory. *ACM Trans. Comput. Log.*, 6(1):61–101, 2005.

17 R. Hasegawa. Categorical data types in parametric polymorphism. *Math. Structures Comput. Sci.*, 4(1):71–109, 1994.

18 W. A. Howard. Assignment of ordinals to terms for primitive recursive functionals of finite type. In A. Kino, J. Myhill, and R. Vesley, editors, *Intuitionism and Proof Theory (Proc. Conf., Buffalo, N.Y., 1968)*, pages 443–458. North-Holland, Amsterdam, 1970.

19 D. Ilik. The exp-log normal form of types: Decomposing extensional equality and representing terms compactly. In *Proceedings of the 44th Symposium on Principles of Programming Languages (POPL 2017)*, pages 387–399. ACM Press, Jan. 2017.

20 B. Jay and N. Ghani. The virtue of eta-expansion. *J. Funct. Programming*, 5(2):135–154, 1995.

21 J.-P. Jouannaud and A. Rubio. Polymorphic higher-order recursive path orderings. *J. ACM*, 54(1):1–48, 2007.

22 Z. Khasidashvili and V. van Oostrom. Context-sensitive conditional expression reduction systems. *Electr. Notes Theor. Comput. Sci.*, 2:167–176, 1995.

23 J. W. Klop. Combinatory reduction systems. Technical report, Mathematical center tracts, 1980.

24 C. Kop. *Higher Order Termination*. PhD thesis, Vrije University, 2012.

25 S. Lindley. Extensional rewriting with sums. In S. Ronchi Della Rocca, editor, *Proceedings of the 8th International Conference on Typed Lambda Calculus and Applications*, volume 664 of *LNCS*, pages 255–271. Springer, 2007.

26 Z. Luo. *Computation and Reasoning*, volume 11 of *International Series of Monographs on Computer Science*. Oxford University Press, 1994.

27 S. Mac Lane. Why commutative diagrams coincide with equivalent proofs. In *Algebraists' Homage: Papers in Ring Theory and Related Topics (New Haven, Conn., 1981)*, volume 13, pages 387–401. Amer. Math. Soc., Providence, R.I., 1982.

28 G. Mints. Closed categories, and proof theory. *Zap. Naučn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI)*, 68:83–114, 145, 1977.

29 G. Mints. Teorija categorii *i* teoria dokazatelstv. I. In *Aktualnye Problemy Logiki i Metodologii Nauky*, pages 252–278. 1979.

30 G. Mints. Proof theory and category theory. In *Selected Papers in Proof Theory*, pages 157–182. Bibliopolis, Naples; North-Holland Publishing Co., Amsterdam, 1992.

31 G. Pottinger. The church rosser theorem for the typed lambda-calculus with surjective pairing. *Notre Dame J. of Formal Logic*, 22(3):264–268, 1981.

32 S. Sarkar. Metatheory of LF extended with dependent pair and unit types. CMU-CS-05-179, School of Computer Science, Carnegie Mellon University, 2005.

33 G. Scherer. Deciding equivalence with sums and the empty type. In *Proceedings of the 44th Symposium on Principles of Programming Languages (POPL 2017)*, pages 374–386. ACM Press, Jan. 2017.

34 K. Schütte. *Proof Theory*, volume 225 of *Grundlehren der Mathematischen Wissenschaften*. Springer-Verlag, Berlin-New York, 1977.

35 Y. Toyama. On the church-rosser property of term rewriting systems. Technical report 17672, NTT ECL, Dec. 1981. In Japanese.

36 V. van Oostrom. Developing developments. *Theor. Comput. Sci.*, 175:159–181, 1997.

37 V. van Oostrom and H. Zantema. Triangulation in rewriting. In A. Tiwari, editor, *Proceedings of the 23rd International Conference on Rewriting Techniques and Applications (RTA'12)*, volume 15 of *LIPIcs*, pages 240–255. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.RTA.2012.240.

38 H. Yokouchi. *On Categorical Models of the Lambda Calculus*. PhD thesis, Tokyo Institute of Technology, 1986.

A SN proof for extension by weakly extensional sum types, by restricted reducibility theorem

Recent study of type-directed η-expansion pay attention to sum types (with permutative conversion) and empty types. See [4, 10, 19, 25, 33], to cite a few. For the first step, we introduce Curien-Di Cosmo-style rewrite rule schemata for weak extensionality for sum types, and then prove the SN.

▶ **Definition 30** $\left(\left(\lambda^{\top,\to,\times,+}\right)'\right)$**.** To the equational theory $\lambda\beta\eta\pi*$, we add *sum types* $\varphi_1+\varphi_2$, *the case distinctions* $\mathrm{d}(t^{\varphi_1+\varphi_2},\ t_1^{\varphi_1\to\psi},\ t_2^{\varphi_2\to\psi})^{\psi}$, *injections* $\left(\mathrm{in}^i_{\varphi_1,\varphi_2}(w^{\varphi_i})\right)^{\varphi_1+\varphi_2}$ $(i=1,2)$, and the following two equational axioms:

$$\mathrm{d}(\mathrm{in}^i_{\varphi_1,\varphi_2}(w),\ t_1,\ t_2) = t_i w, \qquad (i=1,2).$$
$$\mathrm{d}(t,\ \lambda x_1.\,\mathrm{in}^1_{\varphi_1,\varphi_2}(x_1),\ \lambda x_2.\,\mathrm{in}^2_{\varphi_1,\varphi_2}(x_2)) = t.$$

We write $\lambda^{\top,\to,\times,+}$ for the resulting equational theory. We orient the last two axioms:

$$(+.\beta_i) \qquad \mathrm{d}(\mathrm{in}^i_{\varphi_1,\varphi_2}(w),\ t_1,\ t_2) \to t_i w, \qquad (i=1,2).$$
$$(+.\eta) \qquad \mathrm{d}(t,\ \lambda x_1.\,\mathrm{in}^1_{\varphi_1,\varphi_2}(x_1),\ \lambda x_2.\,\mathrm{in}^2_{\varphi_1,\varphi_2}(x_2)) \to t.$$

The rewrite rule schemata $(+.\eta)$ and (g) yields obstructions [2] to CR:

$$\mathrm{d}(y,\ \lambda x_1.\,\mathrm{in}^1_{\varphi_1,\tau}(x_1),\ \lambda x_2.\,\mathrm{in}^2_{\varphi_1,\tau}(*^\tau)) \leftarrow_g \mathrm{d}(y,\ \lambda x_1.\,\mathrm{in}^1_{\varphi_1,\tau}(x_1),\ \lambda x_2.\,\mathrm{in}^2_{\varphi_1,\tau}(x_2)) \to_{+.\eta} y.$$
$$\mathrm{d}(y,\ \lambda x_1.\,\mathrm{in}^1_{\tau,\varphi_2}(*^\tau),\ \lambda x_2.\,\mathrm{in}^2_{\tau,\varphi_2}(x_2)) \leftarrow_g \mathrm{d}(y,\ \lambda x_1.\,\mathrm{in}^1_{\tau,\varphi_2}(x_1),\ \lambda x_2.\,\mathrm{in}^2_{\tau,\varphi_2}(x_2)) \to_{+.\eta} y.$$

Here $\tau \in \mathit{Iso}(\top)$. The set $\mathit{Iso}(\top)$ of "isomorphic to" $\top$ in $\lambda^{\top,\to,\times,+}$ is constructed exactly as that of $\lambda\beta\eta\pi*$ (Definition 1). Below, τ ranges over $\mathit{Iso}(\top)$. The following rule schemata rewrite the leftmost terms of the two obstructions of CR, to the corresponding rightmost terms y.

$$(+.\eta_{top1}) \qquad \mathrm{d}(t,\ \lambda x_1.\,\mathrm{in}^1_{\varphi_1,\tau}(x_1),\ \lambda x_2.\,\mathrm{in}^2_{\varphi_1,\tau}(*^\tau)) \to t,$$
$$(+.\eta_{top2}) \qquad \mathrm{d}(t,\ \lambda x_1.\,\mathrm{in}^1_{\tau,\varphi_2}(*^\tau),\ \lambda x_2.\,\mathrm{in}^2_{\tau,\varphi_2}(x_2)) \to t.$$

Let $\left(\lambda^{\top,\to,\times,+}\right)'$ be the rewriting system consisting of (g), (β), (η), (π_1), (π_2), (SP), (η_{top}), (SP_{top1}), (SP_{top2}), $(+.\beta_1)$, $(+.\beta_2)$, $(+.\eta)$, $(+.\eta_{top1})$, and $(+.\eta_{top2})$.

[2] We avoid saying "critical pairs," since the notion of critical pairs is not so clear in non-left-linear higher-order rewriting systems.

▶ **Definition 31** (Reducibility). The *reducibility* for terms of $\left(\lambda^{\top,\to,\times,+}\right)'$ is defined inductively as in Definition 5 but also with the following clause:

(+) A term $t^{\varphi_1+\varphi_2}$ is *reducible*, if t is neutral and t' is reducible whenever $t \to t'$; or $t \equiv \mathrm{in}^i_{\varphi_1,\varphi_2}(u^{\varphi_i})$ for some $i = 1, 2$ and some reducible u^{φ_i}.

▶ **Definition 32** (Neutral). A term is *neutral* if it is not of the form $\langle u, v\rangle$, $\lambda x.\, v$, $\mathrm{in}^i_{\varphi,\psi}(w)$.

▶ **Lemma 33.** *(CR0), (CR1), (CR2) and (CR3) hold for* $\left(\lambda^{\top,\to,\times,+}\right)'$.

Proof. By induction on the type φ of a given term t. Consider case where $\varphi = \varphi_1 + \varphi_2$.

(CR0) There is no canonical term of type $\varphi_1 + \varphi_2$.

(CR1) The set of reducible terms is the least set satisfying the three clauses $(a), (\times), (\to)$ of Definition 5 and the clause $(+)$ of Definition 31. If we replace 'reducible' with 'SN,' the four clauses hold. So, the set of reducible terms is a subset of the set of SN terms.

(CR2) Suppose t is reducible. If t is neutral, then t' is reducible by Definition 31. Otherwise $t \equiv \mathrm{in}^i_{\varphi_1,\varphi_2}(u^{\varphi_i})$, $t' \equiv \mathrm{in}^i_{\varphi_1,\varphi_2}(u'^{\varphi_i})$, $u^{\varphi_i} \to u'^{\varphi_i}$ for some i and some terms $u_i^{\varphi_i}, u_i'^{\varphi_i}$. By induction hypothesis (CR2) for φ_i, u_i' is reducible. Hence t' is reducible by Definition 31.

(CR3) Immediate from Definition 31.

The other cases are checked as Lemma 6 was proved. ◀

We define $*$-*free* terms, exactly as in Definition 2. Then Lemma 3 holds for $\left(\lambda^{\top,\to,\times,+}\right)'$.

▶ **Lemma 34** (Case). *If t, t_1, and t_2 are reducible. so is* $\mathrm{d}(t,\ t_1,\ t_2)$ *is reducible.*

Proof. By induction on the type ψ of $\mathrm{d}(t^{\varphi_1+\varphi_2},\ t_1^{\varphi_1\to\psi},\ t_2^{\varphi_1\to\psi})$.

ψ is atomic. Let $\mathrm{d}(t,\ t_1,\ t_2) \overset{\Delta}{\to} s$. By (CR1), t, t_1, t_2 are SN. We will verify that s is reducible, by WF induction on

$$\left(\left\{t^{\varphi_1+\varphi_2} \mid\ t^{\varphi_1+\varphi_2} \text{ is reducible}\right\},\ \to\right) \# \left(\left\{t_1^{\varphi_1\to\psi} \mid\ t_1^{\varphi_1\to\psi} \text{ is reducible}\right\},\ \to\right) \# \left(\left\{t_2^{\varphi_1\to\psi} \mid\ t_2^{\varphi_2\to\psi} \text{ is reducible}\right\},\ \to\right) \tag{8}$$

where $\to$ is the rewrite relation. If $\Delta \not\equiv \mathrm{d}(t,\ t_1,\ t_2)$, s is reducible by the WF induction hypothesis. Otherwise $\Delta \equiv \mathrm{d}(t,\ t_1,\ t_2)$. We have three cases, as Δ is $*$-free.

1. $s \equiv *$: Then s is reducible by (CR0).
2. $s \equiv t$ by $(+.\eta)$, $(+.\eta_{top1})$ or $(+.\eta_{top2})$: s is reducible by the premise.
3. Otherwise, $s \equiv t_i w$ and $t \equiv \mathrm{in}^i_{\varphi_1,\varphi_2}(w)$. w is reducible by Definition 31. By the premise, s is reducible.

$\psi = \psi_1 \times \psi_2$ ($\psi_1 \to \psi_2$, resp.) We verify the reducibility of $u :\equiv \pi_i(\mathrm{d}(t,\ t_1,\ t_2))$ ($u :\equiv \mathrm{d}(t,\ t_1,\ t_2)r$ for all reducible r^{ψ_1}, resp.). As t is neutral, we use (CR3) for ψ_i (ψ_2, resp.). Since t, t_1, t_2 (t, t_1, t_2, r, resp.) are all reducible by the premise, they are all SN by (CR1). When $\psi = \psi_1 \times \psi_2$, we proceed by WF induction on the well-founded relation (8). When $\psi = \psi_1 \to \psi_2$, we proceed by WFI $\left((8) \# \left(\left\{r^{\psi_1} \mid\ r^{\psi_1} \text{ is reducible}\right\},\ \to\right)\right)$. Let $u \overset{\Delta}{\to} s$. We have three cases.

1. $\Delta \equiv u$: Then $s \equiv *^{\psi_i}$ ($s \equiv *^{\psi_2}, resp.$), which is reducible by (CR0).
2. $\Delta \equiv \mathrm{d}(t,\ t_1,\ t_2)$, we have two subcases, as Δ is $*$-free.
   a. $s \equiv \pi_i t$ (tr, resp.) by $(+.\eta)$, $(+.\eta_{top1})$ or $(+.\eta_{top2})$: Then s is reducible as t is by the hypothesis.
   b. $s \equiv \pi_i(t_j w)$ ($t_j w r$, resp.) and $t \equiv \mathrm{in}^j_{\varphi_1,\varphi_2}(w)$: Then w is reducible by Definition 31, and so s is reducible by the hypothesis.

3. Otherwise, $\Delta \subseteq t$, $\Delta \subseteq t_1$, $\Delta \subseteq t_2$ or $\Delta \subseteq r$. So, s is reducible by the WF induction hypothesis.

Therefore $\pi_i(\mathrm{d}(t,\ t_1,\ t_2))$ ($\mathrm{d}(t,\ t_1,\ t_2)r$, resp.) is reducible.

$\psi = \varphi_1 + \varphi_2$: Similarly as the case where ψ is atomic. ◀

The following two theorems are proved by exactly the same argument for $(\lambda\beta\eta\pi*)'$.

▶ **Theorem 35** (Restricted Reducibility). *The statement of Theorem 9 holds for* $\left(\lambda^{\top,\to,\times,+}\right)'$.

Proof. The proof is exactly the same as that of Theorem 9, but the case for $\mathrm{in}^i_{\varphi_1,\varphi_2}(t)$ is handled by Definition 31, and the case for $\mathrm{d}(t,\ t_1,\ t_2)$ is by Lemma 34. ◀

▶ **Theorem 36.** *All terms of* $\left(\lambda^{\top,\to,\times,+}\right)'$ *are reducible.*

▶ **Corollary 37.** $\left(\lambda^{\top,\to,\times,+}\right)'$ *satisfies SN.*

It is worth checking whether our approach can ease technicality of the following work on strong sums. Consider an equational theory $\mathcal{N}$ where (1) the type is generated from the unique type constant p, the unit type by means of $+$ ("*sum types*"), $\times$, and $\to$; and (2) the equational axioms are $(\beta), (\eta), (\pi_1), (\pi_2), (SP), (c)$, the usual equational axioms for *case-distinction for sum types* and the *general permutative conversion for sum types*. In [11], Dougherty and Subrahmanyam employed $\to_{\overline{\eta SP}}$ to prove "An equation is provable in $\mathcal{N}$, if and only if it is true in *the* set-theoretic model with the unique atomic type p interpreted as an infinite set." In [25], Lindley provided an SN reduction system based on $\to_{\overline{\eta SP}}$ and proved the "CR modulo" in order to decide the equational theory $\mathcal{N}$. A decision procedure for the extensional typed λ-calculus with function types, product types, strong sum types, the terminal types and empty types is given by Scherer [33] based on focusing, and by Balat-Di Cosmo-Fiore [4] based on normalization by evaluation with turning non-standard permutative conversions into an equivalence relation.

Improving Rewriting Induction Approach for Proving Ground Confluence*

Takahito Aoto[1], Yoshihito Toyama[2], and Yuta Kimura[3]

1 **Faculty of Engineering, Niigata University, Niigata, Japan**
aoto@ie.niigata-u.ac.jp
2 **RIEC, Tohoku University, Sendai, Miyagi, Japan**
toyama@riec.tohoku.ac.jp
3 **Faculty of Engineering, Niigata University, Niigata, Japan**
kimura@nue.ie.niigata-u.ac.jp

Abstract

In (Aoto&Toyama, FSCD 2016), a method to prove ground confluence of many-sorted term rewriting systems based on rewriting induction is given. In this paper, we give several methods that add wider flexibility to the rewriting induction approach for proving ground confluence. Firstly, we give a method to deal with the case in which suitable rules are not presented in the input system. Our idea is to construct additional rewrite rules that supplement or replace existing rules in order to obtain a set of rules that is adequate for applying rewriting induction. Secondly, we give a method to deal with non-orientable constructor rules. This is accomplished by extending the inference system of rewriting induction and giving a sufficient criterion for the correctness of the system. Thirdly, we give a method to deal with disproving ground confluence. The presented methods are implemented in our ground confluence prover AGCP and experiments are reported. Our experiments reveal the presented methods are effective to deal with problems for which state-of-the-art ground confluence provers can not handle.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems, I.2.3 Deduction and Theorem Proving

Keywords and phrases Ground Confluence, Rewriting Induction, Non-Orientable Equations, Term Rewriting Systems

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.7

1 Introduction

Confluence is an important property of computational models having non-deterministic computations. In term rewriting, confluence has caught attention many years. Recent interest has been shifted toward the automated methods for proving confluence. Thus, many confluence tools have been developed in past years, and efforts to have yearly confluence competitions continues[1]. Confluence property considered on the set of ground terms is called *ground confluence*. Confluence implies ground confluence, but not vice versa. In equational logic, ground terms are concerned with the validity on the initial models (inductive validity), and thus, it arises when studying in topics such as inductive theorem proving and correct program transformations, which is closely related to inductive validity.

* This work is partially supported by JSPS KAKENHI No. 15K00003.
[1] http://coco.nue.riec.tohoku.ac.jp

Ground confluence of many-sorted term rewriting systems (TRSs, for short) has been studied in e.g. [4, 5, 8, 9]. Recently, two automated tools for ground confluence appeared [2, 12] – one tool is based on rewriting induction [7, 13], which is originally a method for proving inductive theorems, and the other is based on tree automata technique. However, both approaches have some limitations – in rewriting induction approach, the input TRS needs to have a quasi-reducible terminating set of rewrite rules, since the method is based on noetherian induction using reduction order of terminating rewriting systems; in tree automata approach, the input TRS needs to be left-linear right-ground.

In this paper, we give several methods that add wider flexibility to the rewriting induction approach for proving ground confluence. Firstly, we give a method to deal with the case in which suitable rewrite rules are not presented in the input system. Our idea is to construct additional rewrite rules that supplement or replace existing rules. This approach also provides a way to deal with some kind of non-orientable rewrite rules. But it turns out the approach is not capable of some rules, e.g. those for adding flexibility of the data structures. To deal also with such rules, we next extend the inference system of rewriting induction and give a sufficient criterion for the correctness of the system. The last ingredient of the paper is about disproving ground confluence – we design an inference rule of rewriting induction for refuting ground confluence in the course of rewriting induction derivation. All the presented methods are implemented in our ground confluence prover AGCP and experiments are reported.

The rest of the paper is organized as follows. Section 2 fixes notations used in this paper, and presents basic backgrounds on ground confluence proving based on rewriting induction. In Section 3, we present a method to find and construct a suitable defining rules and a method to replace some non-orientable constructor rules, for the rewriting induction to work. In Section 4, we give an extension of rewriting induction system for bounded ground convertibility that can also deals with non-orientable rewrite rules; and then, this rewriting induction system is applied to obtain a new ground confluence criterion. Section 5 deals with proving ground non-confluence. Implementation and experiments of these methods are reported in Section 6. Section 7 concludes.

2 Preliminaries

We assume basic familiarity with term rewriting (e.g. [15]).

For a quasi-order $\succsim$, its strict part and equivalent part are denoted by $\succ$ and $\approx$, respectively. The reflexive (reflexive transitive, symmetric, equivalent) closure of a relation $\to$ is denoted by $\to^=$ (resp. $\to^*$, $\leftrightarrow$, $\leftrightarrow^*$) and n-fold composition by $\to^n$ ($n \geq 0$). We write $\overset{\succ}{\to}$ ($\overset{\approx}{\to}$, $\overset{\succ}{\to}$) to denote $\succsim \cap \to$ (resp. $\approx \cap \to$, $\succ \cap \to$). Let $\to$ be a relation and $\succsim$ a quasi-order on a set. Elements x and y are *bounded convertible* if $x = x_0 \leftrightarrow x_1 \leftrightarrow \cdots \leftrightarrow x_n = y$ for some $x_0, \ldots, x_n$ such that $x \succsim x_i$ or $y \succsim x_i$ $(0 \leq i \leq n)$; we write $x \overset{*}{\leftrightarrow}_{\succsim} y$ if x and y are bounded convertible. We use $\circ$ for the function composition. A relation $\to$ is *confluent* if ${}^*\!\leftarrow \circ \to^* \subseteq \to^* \circ {}^*\!\leftarrow$.

A *many-sorted signature* is given by a non-empty set $\mathcal{S}$ of sorts and a set $\mathcal{F}$ of many-sorted *function symbols*. We use $\mathcal{V}$ to denote the set of many-sorted variables. We will not explicitly state the sort information if it is understood from the context. The sets of function symbols and variables in a term t are denoted by $\mathcal{F}(t)$ and $\mathcal{V}(t)$, respectively. A term t is *ground* if $\mathcal{V}(t) = \emptyset$; it is *linear* if any variable occurs at most once in t. We use $\uplus$ for the disjoint union. We consider a partition of function symbols $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$; function symbols in $\mathcal{D}$ are called *defined symbols* and those in $\mathcal{C}$ are *constructors*. For any $\mathcal{G} \subseteq \mathcal{F}$, the set of (ground) terms over function symbols $\mathcal{G}$ is denoted by $\mathrm{T}(\mathcal{G}, \mathcal{V})$ (resp. $\mathrm{T}(\mathcal{G})$). Terms in $\mathrm{T}(\mathcal{C}, \mathcal{V})$ are called *constructor terms*. A term t is *basic* if it has the form $f(c_1, \ldots, c_n)$ for some $f \in \mathcal{D}$ and $c_1, \ldots, c_n \in \mathrm{T}(\mathcal{C}, \mathcal{V})$. The set of basic subterms of t is denoted by $\mathcal{B}(t)$.

A *substitution* is a sort-preserving mapping $\theta : \mathcal{V} \to \mathrm{T}(\mathcal{F}, \mathcal{V})$ such that $\mathrm{dom}(\theta) = \{x \in \mathcal{V} \mid \theta(x) \neq x\}$ is finite. A substitution θ is *ground* (*constructor*, *ground constructor*) if $\theta(x) \in \mathrm{T}(\mathcal{F})$ (resp. $\mathrm{T}(\mathcal{C}, \mathcal{V})$, $\mathrm{T}(\mathcal{C})$) for all $x \in \mathrm{dom}(\theta)$; we assume $\mathrm{dom}(\theta) \subseteq \mathcal{V}(t)$ when we write $t\theta$ for a substitution θ. A set L of terms *covers* a term t if for any ground constructor substitution σ_{gc}, there exists $l \in L$ such that $\exists\theta.\, t\sigma_{gc} = l\theta$.

For a *position* p in a term t, $t(p)$ denotes the function symbol or variable at p and $t|_p$ denotes the subterm at position p. We use ϵ for the *root* position. A *context* $C[\,]$ is a term with a sorted hole, and $C[t]$ denotes the term obtained by filling the hole with a term t of the same sort. A relation is a *rewrite relation* if it closed under contexts and substitutions. A *rewrite quasi-order* is a quasi-order whose strict part and equivalent part are rewrite relations. A rewrite quasi-order is a *reduction quasi-order* if its strict part is well-founded.

For any equation $s \doteq t$, we assume s and t have the same sort. (Indirected) equations $s \doteq t$ and $t \doteq s$ are identified. A set $\{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ of equations is (ambiguously) abbreviated as $\{s_i \doteq t_i\}_i$ when no confusion arises. A *directed* equation is denoted by $s \to t$. For a set E of directed equations, we put $E^{\leftrightarrow} = \{l \doteq r \mid l \to r \in E\}$. For a set E of equations (directed equations) the smallest rewrite relation containing E is denoted by $\leftrightarrow_E$ (resp. $\to_E$). A directed equation $l \to r$ subjected to $l \notin \mathcal{V}$ and $\mathcal{V}(l) \supseteq \mathcal{V}(r)$ is a *rewrite rule*. A (many-sorted) *term rewriting system* (*TRS* for short) is a finite set of rewrite rules. A TRS $\mathcal{R}$ is *left-linear* if l is a linear term for all $l \to r \in \mathcal{R}$; it is *variable preserving* if $\mathcal{V}(l) = \mathcal{V}(r)$ for all $l \to r \in \mathcal{R}$. A TRS $\mathcal{R}$ is *(ground) confluent* if $\to_{\mathcal{R}}$ is confluent on $\mathrm{T}(\mathcal{F}, \mathcal{V})$ ($\mathrm{T}(\mathcal{F})$, respectively); $\mathcal{R}$ is *terminating* if $\to_{\mathcal{R}}$ is well-founded; $\mathcal{R}$ is *sufficiently complete* if for any $t_g \in \mathrm{T}(\mathcal{F})$ there exists $u_g \in \mathrm{T}(\mathcal{C})$ such that $t_g \to_{\mathcal{R}}^* u_g$. The set of rules $l \to r \in \mathcal{R}$ satisfying $l(\epsilon) \notin \mathcal{D}$ is denoted by $\mathcal{R}_c$ and called the set of *constructor rules*. For a rewrite quasi-order $\succsim$, the strict part of $\mathcal{R}$ is given by $\mathcal{R}^{\succ} = \{l \to r \in \mathcal{R} \mid l \succ r\}$ and the equivalent part of $\mathcal{R}$ by $\mathcal{R}^{\approx} = \{l \to r \in \mathcal{R} \mid l \approx r\}$. Note, as easily seen, $\mathcal{R}^{\approx}$ is variable preserving.

A *most general unifier* of terms s and t is denoted by $\mathrm{mgu}(s, t)$. Let $l_1 \to r_1, l_2 \to r_2$ be rewrite rules whose variables are w.l.o.g. renamed so that $\mathcal{V}(l_1) \cap \mathcal{V}(l_2) = \emptyset$. We say $l_1 \to r_1$ *overlaps on* $l_2 \to r_2$ when l_1 and non-variable subterm $l_2|_p$ of l_2 unify. The overlap arises a *critical pair* $l_2[r_1]_p\theta \mathrel{\widehat{\leftarrow}} r_2\theta$, where $\theta = \mathrm{mgu}(l_1, l_2|_p)$. The set of critical pairs arises from overlaps of rules in $\mathcal{R}$ on rules in $\mathcal{S}$ is denoted by $\mathrm{CP}(\mathcal{R}, \mathcal{S})$. We put $\mathrm{CP}(\mathcal{R}) = \mathrm{CP}(\mathcal{R}, \mathcal{R})$ and regard it as the set of equations induced from the critical pairs.

We write $\mathcal{R} \otimes f = \{f(\ldots, x_{i-1}, l, x_{i+1}, \ldots) \to f(\ldots, x_{i-1}, r, x_{i+1}, \ldots) \mid 1 \leq i \leq n, l \to r \in \mathcal{R}\}$, where $x_1, \ldots, x_n$ are pairwise distinct variables not in l. Let $\mathrm{LHS}(\mathcal{R}) = \{l \mid l \to r \in \mathcal{R}\}$ and $\mathrm{LHS}(f, \mathcal{R}) = \{l \mid l(\epsilon) = f, l \to r \in \mathcal{R}\}$. A TRS $\mathcal{R}$ is said to be a *strongly quasi-reducible* (w.r.t. $\mathcal{D}$) if $\mathrm{LHS}(\mathcal{R}_c \otimes f) \cup \mathrm{LHS}(f, \mathcal{R})$ covers $f(x_1, \ldots, x_n)$ for each $f \in \mathcal{D}$. It easily follows from the definition that any strongly quasi-reducible TRS is quasi-reducible TRS [2]. Thus, any terminating and strongly quasi-reducible TRS is sufficiently complete [10].

In [2], the first two authors give a system of rewriting induction for proving ground confluence of TRSs – its inference rules are given in Figure 1. In *Expand* rule, we put $\mathrm{Expd}_u(s, t) = \{C[r]\mu \to t\mu \mid \mu = \mathrm{mgu}(l, u), l \to r \in (\mathcal{R} \setminus \mathcal{R}_c) \cup (\mathcal{R}_c \otimes f)\}$, where $s = C[u]$ and $u(\epsilon) = f$. We write $\langle E, H\rangle \rightsquigarrow_{\mathcal{R},\mathcal{D},\succsim} \langle E', H'\rangle$ (or $\langle E, H\rangle \rightsquigarrow \langle E', H'\rangle$ if no confusion arises) when $\langle E', H'\rangle$ is obtained from $\langle E, H\rangle$ by an inference rule.

Let $\succsim$ be a quasi-order on terms. An equation $s \doteq t$ is *bounded ground convertible* (by $\mathcal{R}$ w.r.t. $\succsim$) if all ground instantiation $s\theta_g$ and $t\theta_g$ are bounded convertible for the relation $\to_{\mathcal{R}}$ and quasi-order $\succsim$. Then we have the following proposition.

▶ **Proposition 1** ([2]). *Let $\succsim$ be a reduction quasi-order and $\mathcal{R}$ a strongly quasi-reducible TRS w.r.t. $\mathcal{D}$ such that $\mathcal{R} \subseteq \succ$. (1) If $\langle E, \emptyset\rangle \rightsquigarrow^*_{\mathcal{R},\mathcal{D},\succsim} \langle \emptyset, H\rangle$ for some H, then E is bounded ground convertible by $\mathcal{R}$ w.r.t. $\succsim$. (2) If $\langle \mathrm{CP}(\mathcal{R}), \emptyset\rangle \rightsquigarrow^*_{\mathcal{R},\mathcal{D},\succsim} \langle \emptyset, H\rangle$ for some H, then $\mathcal{R}$ is ground confluent.*

Expand

$$\frac{\langle E \uplus \{s \doteq t\},\ H\rangle}{\langle E \cup \{s'_i \doteq t_i\}_i,\ H \cup \{s \to t\}\rangle} \quad \begin{array}{l} u \in \mathcal{B}(s), \{s_i \to t_i\}_i = \mathrm{Expd}_u(s,t), \\ s_i \overset{\succ}{\rightsquigarrow}^*_{H\leftrightarrow} s'_i \end{array}$$

Simplify

$$\frac{\langle E \uplus \{s \doteq t\},\ H\rangle}{\langle E \cup \{s' \doteq t\},\ H\rangle} \quad s \overset{\succ}{\to}_{\mathcal{R}\cup H} \circ \overset{\succ}{\rightsquigarrow}^*_{H\leftrightarrow} s'$$

Delete

$$\frac{\langle E \uplus \{s \doteq t\},\ H\rangle}{\langle E,\ H\rangle} \quad s \to^{\equiv}_{H} t$$

Figure 1 Inference rules of rewriting induction for ground confluence proof.

An equation $s \doteq t$ is an *inductive theorem of* $\mathcal{R}$ if $s\theta_g \leftrightarrow^*_{\mathcal{R}} t\theta_g$ for any ground substitution θ_g. We write $\mathcal{R} \models_{ind} E$ for a set E of equations if any equation $s \doteq t \in E$ is an inductive theorem of $\mathcal{R}$. Clearly, if an equation is bounded ground convertible by $\mathcal{R}$, then it is an inductive theorem of $\mathcal{R}$. In practice, it is often useful to incorporate the following easily obtained property into ground confluence proofs by rewriting induction:

▶ **Proposition 2** ([2]). *Let $\mathcal{R}$ be a TRS. Suppose $\mathcal{R}_0 \subseteq \mathcal{R}$ is ground confluent. If $\mathcal{R}_0 \models_{ind} \mathcal{R} \setminus \mathcal{R}_0$ then $\mathcal{R}$ is ground confluent.*

All in all, the procedure for ground confluence proof is described as follows:

Basic GCR Procedure

Input: many-sorted TRS $\mathcal{R}$
Output: `YES` or `MAYBE`

1. Compute (possibly multiple) candidates for the partition $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ of function symbols.
2. Compute (possibly multiple) candidates for strongly quasi-reducible $\mathcal{R}_0 \subseteq \mathcal{R}$.
3. Find a reduction quasi-order $\succsim$ such that $\mathcal{R}_0 \subseteq \succ$.
4. Run rewriting induction for deriving $\langle \mathrm{CP}(\mathcal{R}_0) \cup (\mathcal{R} \setminus \mathcal{R}_0), \emptyset\rangle \overset{*}{\rightsquigarrow}_{\mathcal{R}_0,\mathcal{D},\succsim} \langle \emptyset, H\rangle$ for some H.
5. Return `YES` if it succeeds in step 4, otherwise `MAYBE`.

---

3 Rule Complementation and Instantiation

For our basic GCR procedure to work, it is required that there exists strongly quasi-reducible and terminating subset $\mathcal{R}_0 \subseteq \mathcal{R}$. Experiments in [2], however, reveal that there are cases that there does not exist such an $\mathcal{R}_0$.

▶ **Example 3.** Let $\mathcal{F} = \{\mathsf{eq} : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Bool},\ \mathsf{neq} : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Bool},\ \mathsf{not} : \mathsf{Bool} \to \mathsf{Bool},\ \mathsf{s} : \mathsf{Nat} \to \mathsf{Nat},\ \mathsf{0} : \mathsf{Nat},\ \mathsf{true} : \mathsf{Bool},\ \mathsf{false} : \mathsf{Bool}\}$ and

$$\mathcal{R} = \left\{ \begin{array}{llll} \mathsf{eq}(\mathsf{0},\mathsf{0}) & \to \mathsf{true} & (a) & \mathsf{eq}(\mathsf{0},\mathsf{s}(y)) \to \mathsf{false} \quad (b) \\ \mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) & \to \mathsf{eq}(x,y) & (c) & \mathsf{eq}(x,y) \to \mathsf{eq}(y,x) \quad (d) \\ \mathsf{not}(\mathsf{true}) & \to \mathsf{false} & (e) & \mathsf{not}(\mathsf{false}) \to \mathsf{true} \quad (f) \\ \mathsf{neq}(x,y) & \to \mathsf{not}(\mathsf{eq}(x,y)) & (g) & \mathsf{neq}(x,\mathsf{s}(x)) \to \mathsf{true} \quad (h) \end{array} \right\}$$

Then, only possible reduction $\mathsf{eq}(\mathsf{s}(\mathsf{0}),\mathsf{0}) \to^*_{\mathcal{R}} \mathsf{false}$ has the following form: $\mathsf{eq}(\mathsf{s}(\mathsf{0}),\mathsf{0}) \to_{\{(d)\}} \mathsf{eq}(\mathsf{0},\mathsf{s}(\mathsf{0})) \to_{\{(b)\}} \mathsf{false}$. Hence, in order to make $\mathcal{R}_0$ strong quasi-reducible, one needs

$(d) \in \mathcal{R}_0$. But having $(d) \in \mathcal{R}_0$ makes $\mathcal{R}_0$ non-terminating. Thus, there exists no quasi-reducible and terminating subset $\mathcal{R}_0$ of $\mathcal{R}$. A natural candidate of $\mathcal{R}_0$ here would be

$$\mathcal{R}_0' = \left\{ \begin{array}{llllllll} \mathsf{eq}(0,0) & \to & \mathsf{true} & (a) & \mathsf{eq}(0,\mathsf{s}(y)) & \to & \mathsf{false} & (b) \\ \mathsf{eq}(\mathsf{s}(x),0) & \to & \mathsf{false} & (b') & \mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}(x,y) & (c) \end{array} \right\} \cup \{(e),(f),(g)\}$$

Indeed, the rewrite rule (b') is equationally valid as: $\mathsf{eq}(\mathsf{s}(x),0) \to_{\{(d)\}} \mathsf{eq}(0,\mathsf{s}(x)) \to_{\{(b)\}} \mathsf{false}$. However, the rewrite rule (b') is not contained in $\mathcal{R}$. Let $\mathcal{R}' = \mathcal{R} \cup \{(b')\}$. Then, we have $\to_{\mathcal{R}} \subseteq \to_{\mathcal{R}'} \subseteq \to_{\mathcal{R}}^*$, and thus, ground confluence of $\mathcal{R}'$ implies that of $\mathcal{R}$. Indeed, $\mathrm{CP}(\mathcal{R}_0') = \emptyset$ and $\mathcal{R}_0' \models_{ind} \{(b),(h)\}$ is obtained in our basic GCR procedure, and thus ground confluence of $\mathcal{R}'$ can be shown. The key part of this procedure is to find the lacking pattern $\mathsf{eq}(\mathsf{s}(x),0)$ and obtain a suitable rewrite rule (b').

In the previous example, we supplement rewrite rules. There are cases that we need to replace the rewrite rules, instead of supplementing new ones.

▶ **Example 4.** Let $\mathcal{F} = \{\mathsf{zero} : \mathsf{Nat} \to \mathsf{Bool},\ \mathsf{if} : \mathsf{Bool} \times \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat},\ + : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat},\ - : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat},\ * : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat},\ \mathsf{fact} : \mathsf{Nat} \to \mathsf{Nat},\ \mathsf{s} : \mathsf{Nat} \to \mathsf{Nat},\ 0 : \mathsf{Nat},\ \mathsf{true} : \mathsf{Bool},\ \mathsf{false} : \mathsf{Bool}\}$ and

$$\mathcal{R} = \left\{ \begin{array}{llllll} \mathsf{zero}(0) & \to & \mathsf{true} & \mathsf{if}(\mathsf{true},y,z) & \to & y \\ \mathsf{zero}(\mathsf{s}(x)) & \to & \mathsf{false} & \mathsf{if}(\mathsf{false},y,z) & \to & z \\ +(0,y) & \to & y & -(0,y) & \to & 0 \\ +(\mathsf{s}(x),y) & \to & \mathsf{s}(+(x,y)) & -(x,0) & \to & x \\ +(x,\mathsf{s}(y)) & \to & \mathsf{s}(+(y,x)) & -(\mathsf{s}(x),\mathsf{s}(y)) & \to & -(x,y) \\ *(0,y) & \to & 0 & *(\mathsf{s}(x),y) & \to & +(*(x,y),y) \\ \mathsf{fact}(x) & \to & \mathsf{if}(\mathsf{zero}(x),\mathsf{s}(0),*(x,\mathsf{fact}(-(x,\mathsf{s}(0))))) & & & (a) \end{array} \right\}$$

The rewrite rule (a) is the only rule that defines the function fact. Thus, we need to have $(a) \in \mathcal{R}_0$, but the lhs of (a) embeds in the rhs, and thus there is no reduction quasi-order $\succsim$ such that $(a) \in \succ$. To make the rewriting induction work, we replace the rule (a) with the following two rules that inductively defines the function fact:

$$\left\{ \begin{array}{llll} \mathsf{fact}(0) & \to & \mathsf{s}(0) & (a') \\ \mathsf{fact}(\mathsf{s}(x)) & \to & *(\mathsf{s}(x),\mathsf{fact}(x)) & (a'') \end{array} \right\}$$

These new rules are obtained by:

$$\mathsf{fact}(0) \to_{\mathcal{R}} \mathsf{if}(\mathsf{zero}(0),\mathsf{s}(0),\ldots) \to_{\mathcal{R}} \mathsf{if}(\mathsf{true},\mathsf{s}(0),\ldots) \to_{\mathcal{R}} \mathsf{s}(0)$$
$$\mathsf{fact}(\mathsf{s}(x)) \to_{\mathcal{R}} \mathsf{if}(\mathsf{zero}(\mathsf{s}(x)),\ldots,*(\mathsf{s}(x),\mathsf{fact}(-(\mathsf{s}(x),\mathsf{s}(0))))) \to_{\mathcal{R}}^* *(\mathsf{s}(x),\mathsf{fact}(x))$$

In this example, we need to construct a pattern $\{\mathsf{fact}(0),\mathsf{fact}(\mathsf{s}(x))\}$ that covers $\mathsf{fact}(x)$.

Such lacking patterns can be characterized by the notion of complement of patterns: A finite set P of basic terms is called a *pattern*. Intuitively, the set P can be regarded as expressing a set of ground terms given as $\mathrm{Inst}(P) = \{p\sigma_{gc} \mid p \in P, \sigma_{gc} : \mathcal{V} \to \mathrm{T}(\mathcal{C})\}$. $\mathrm{T_B}(\mathcal{D},\mathcal{C})$ denotes the set of ground basic terms over $\mathcal{D}$ and $\mathcal{C}$. A finite set Q of terms is said to be a *complement* (w.r.t. $\mathrm{T_B}(\mathcal{D},\mathcal{C})$) of P if $\mathrm{Inst}(P) \uplus \mathrm{Inst}(Q) = \mathrm{T_B}(\mathcal{D},\mathcal{C})$. We (ambiguously) denote Q as $\mathrm{T_B}(\mathcal{D},\mathcal{C}) \ominus P$.

▶ **Example 5.** Let $\mathcal{D} = \{\mathsf{eq}\}$, $\mathcal{C} = \{\mathsf{s},0,\mathsf{true},\mathsf{false}\}$ and $P = \{\mathsf{eq}(0,0),\mathsf{eq}(0,\mathsf{s}(y)),\mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y))\}$. Then $\mathrm{T_B}(\mathcal{D},\mathcal{C}) \ominus P = \{\mathsf{eq}(\mathsf{s}(x),0)\}$.

A pattern P is *linear* if so are all its elements. Theorem 1 of [11] gives an algorithm to compute Q from P (complementation algorithm) for any linear pattern P, and we use it in the following procedure.

▶ **Definition 6** (Rule Complementation Procedure)**.**

1. For each $f \in \mathcal{D}$,
   - **a.** take $\mathcal{S}_f \subseteq \{l \to r \in \mathcal{R} \mid l(\epsilon) = f\}$ such that $\mathrm{LHS}(\mathcal{S}_f)$ is a linear pattern, and
   - **b.** take $l_1, \ldots, l_n$ such that $\{l_1, \ldots, l_n\} = \mathrm{T_B}(\{f\}, \mathcal{C}) \ominus \mathrm{LHS}(\mathcal{S}_f)$ and $r_1, \ldots, r_n$ such that $l_i \to_{\mathcal{R}}^* r_i$ for each i, and let $\mathcal{S}'_f = \mathcal{S}_f \cup \{l_i \to r_i \mid 1 \le i \le n\}$.
2. Finally, put $\mathcal{R}_1 = \mathcal{R}_c \cup \bigcup_{f \in \mathcal{D}} \mathcal{S}'_f$.

▶ **Theorem 7.** *Suppose $\mathcal{R}_1$ is obtained from $\mathcal{R}$ by the rule complementation procedure. If $\mathcal{R}_1$ is ground confluent and $\mathcal{R}_1 \models_{ind} \mathcal{R} \setminus \mathcal{R}_1$, then $\mathcal{R}$ is ground confluent.*

Proof. Suppose $\mathcal{R}_1$ is ground confluent and $\mathcal{R}_1 \models_{ind} \mathcal{R} \setminus \mathcal{R}_1$. Then $\mathcal{R}_1 \cup \mathcal{R}$ is ground confluent by Proposition 2. From the procedure, we have $\to_{\mathcal{R}} \subseteq \to_{\mathcal{R}_1 \cup \mathcal{R}} \subseteq \to_{\mathcal{R}}^*$. Thus, ground confluence of $\mathcal{R}$ follows. ◀

▶ **Example 8.** Let $\mathcal{R}$ be a TRS given in Example 4. Take $\mathcal{S}_{\mathsf{fact}} = \emptyset$, and one can put $P = \mathrm{T_B}(\{\mathsf{fact}\}, \mathcal{C}) \ominus \emptyset = \{\mathsf{fact}(\mathsf{0}), \mathsf{fact}(\mathsf{s}(x))\}$. By $\mathsf{fact}(\mathsf{0}) \to_{\mathcal{R}}^* \mathsf{0}$ and $\mathsf{fact}(\mathsf{s}(x)) \to_{\mathcal{R}}^* *(\mathsf{s}(x), \mathsf{fact}(x))$, one gets $\mathcal{R}_1 = \mathcal{R} \setminus \{(a)\} \cup \{(a'), (a'')\}$. Then one successfully proves that $\mathcal{R}_1$ is ground confluent and $\mathcal{R}_1 \models_{ind} \{(a)\}$. Thus, it follows that $\mathcal{R}$ is ground confluent.

The following example shows the condition $\mathcal{R}_1 \models_{ind} \mathcal{R} \setminus \mathcal{R}_1$ in Theorem 7 can not be dropped.

▶ **Example 9.** Let $\mathcal{F} = \{\mathsf{zero} : \mathsf{Nat} \to \mathsf{Bool},\ \mathsf{if} : \mathsf{Bool} \times \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat},\ \mathsf{f} : \mathsf{Nat} \to \mathsf{Nat},\ \mathsf{s} : \mathsf{Nat} \to \mathsf{Nat},\ \mathsf{0} : \mathsf{Nat},\ \mathsf{true} : \mathsf{Bool},\ \mathsf{false} : \mathsf{Bool}\}$ and

$$\mathcal{R} = \left\{ \begin{array}{llllll} \mathsf{zero}(\mathsf{0}) & \to & \mathsf{true} & \mathsf{if}(\mathsf{true}, y, z) & \to & y \\ \mathsf{zero}(\mathsf{s}(x)) & \to & \mathsf{false} & \mathsf{if}(\mathsf{false}, y, z) & \to & z \\ \mathsf{f}(\mathsf{0}) & \to & \mathsf{0} & & & (a) \\ \mathsf{f}(x) & \to & \mathsf{if}(\mathsf{zero}(x), \mathsf{s}(\mathsf{0}), \mathsf{0}) & & & (b) \end{array} \right\}$$

Take $\mathcal{S}_{\mathsf{f}} = \{(a)\}$ and $\mathrm{T_B}(\{\mathsf{f}\}, \mathcal{C}) \ominus \mathrm{LHS}(\mathcal{S}_{\mathsf{f}}) = \{\mathsf{f}(\mathsf{s}(x))\}$. By $\mathsf{f}(\mathsf{s}(x)) \to \mathsf{if}(\mathsf{zero}(\mathsf{s}(x)), \mathsf{s}(\mathsf{0}), \mathsf{0}) \to \mathsf{if}(\mathsf{false}, \mathsf{s}(\mathsf{0}), \mathsf{0}) \to \mathsf{0}$, one gets $\mathcal{S}'_{\mathsf{f}} = \{(a)\} \cup \{\mathsf{f}(\mathsf{s}(x)) \to \mathsf{0}\}$. Now as $\mathcal{R}_1$ is orthogonal, $\mathcal{R}_1$ is ground confluent. On the other hand, we have $\mathsf{0} \leftarrow_{\mathcal{R}} \mathsf{f}(\mathsf{0}) \to_{\mathcal{R}} \mathsf{if}(\mathsf{zero}(\mathsf{0}), \mathsf{s}(\mathsf{0}), \mathsf{0}) \to_{\mathcal{R}} \mathsf{if}(\mathsf{true}, \mathsf{s}(\mathsf{0}), \mathsf{0}) \to_{\mathcal{R}} \mathsf{s}(\mathsf{0})$, and thus $\mathcal{R}_1$ is not ground confluent. Note here $\mathcal{R}_1 \not\models_{ind} \{(b)\}$.

Instantiation technique is also useful for dealing with non-orientable constructor rules. The following example illustrates this.

▶ **Example 10** (COPS #74)**.** Let $\mathcal{R}$ be a (uni-sorted) TRS as follows:

$$\left\{ \begin{array}{llllllllll} \mathsf{a} & \to & \mathsf{c} & (a) & \mathsf{b} & \to & \mathsf{c} & (b) & \mathsf{f}(\mathsf{a}, \mathsf{b}) & \to & \mathsf{d} & (c) \\ \mathsf{f}(x, \mathsf{c}) & \to & \mathsf{f}(\mathsf{c}, \mathsf{c}) & (d) & \mathsf{f}(\mathsf{c}, x) & \to & \mathsf{f}(\mathsf{c}, \mathsf{c}) & (e) & & & & \\ \mathsf{d} & \to & \mathsf{f}(\mathsf{a}, \mathsf{c}) & (f) & \mathsf{d} & \to & \mathsf{f}(\mathsf{c}, \mathsf{b}) & (g) & & & & \end{array} \right.$$

Let $\mathcal{C} = \{\mathsf{f}, \mathsf{c}\}$. Take $\mathcal{R}_c = \{(d), (e)\}$, $\mathcal{R}_0 = \{(a), (b), (g)\} \cup \mathcal{R}_c$ and $\mathcal{R} \setminus \mathcal{R}_0 = \{(c), (f)\}$. Then $\mathcal{R}_0$ is strongly quasi-reducible. However, $\mathcal{R}_0 \subseteq \succ$ does no hold because $\mathsf{f}(x, \mathsf{c}) \not\succ \mathsf{f}(\mathsf{c}, \mathsf{c})$ as well as $\mathsf{f}(\mathsf{c}, x) \not\succ \mathsf{f}(\mathsf{c}, \mathsf{c})$.

Instantiation technique can be used as follows. First, add some instantiation $\mathcal{R}'_c$ of rewrite rules (d) and (e) and replace partitions as follows:

$$\mathcal{R}'_c = \left\{ \begin{array}{lcll} \mathsf{f}(\mathsf{c},\mathsf{c}) & \to & \mathsf{f}(\mathsf{c},\mathsf{c}) & (h) \\ \mathsf{f}(\mathsf{f}(x,y),\mathsf{c}) & \to & \mathsf{f}(\mathsf{c},\mathsf{c}) & (h') \\ \mathsf{f}(\mathsf{c},\mathsf{f}(x,y)) & \to & \mathsf{f}(\mathsf{c},\mathsf{c}) & (h'') \end{array} \right\}, \qquad \begin{array}{l} \mathcal{R}'_0 = \{(a),(b),(g)\} \cup \mathcal{R}'_c \\ \mathcal{R} \setminus \mathcal{R}'_0 = \{(c),(d),(e),(f)\} \end{array}$$

Because $\mathcal{C} = \{\mathsf{f},\mathsf{c}\}$, $\mathsf{f}(\mathsf{c},x) \doteq \mathsf{f}(\mathsf{c},\mathsf{c})$ are inductive theorems of $\mathcal{R}'_0$. Thus, $\xrightarrow{*}_{\mathcal{R}}$ and $\xrightarrow{*}_{\mathcal{R}\cup\mathcal{R}'_c}$ coincide on ground terms. Evidently, one can further remove trivial rule (h), and thus one can use $\mathcal{R}''_0 = \mathcal{R}'_0 \setminus \{(h)\} \subseteq \succ$. Then the ground confluence of $\mathcal{R}''_0$ can be shown, and from $\mathcal{R}''_0 \models_{ind} \{(c),(d),(e),(f)\}$, one can conclude ground confluence of $\mathcal{R}$.

Now we formalize this idea. A substitution σ is linear if $\sigma(x)$ is linear for all $x \in \mathrm{dom}(\sigma)$ and $\mathcal{V}(\sigma(x)) \cap \mathcal{V}(\sigma(y)) = \emptyset$ for any distinct $x, y \in \mathrm{dom}(\sigma)$. A complement of a linear constructor substitution σ is a set of linear constructor substitutions $\mathrm{Comp}(\sigma)$ such that for any term t and ground constructor substitution θ_{gc}, there exists $\rho \in \mathrm{Comp}(\sigma) \cup \{\sigma\}$ such that $t\theta_g$ is an instance of $t\rho$. Definition 11 of [11] gives an algorithm to compute a complement $\mathrm{Comp}(\sigma)$ of a linear constructor substitution σ, and we use it in the following procedure.

▶ **Definition 11** (Rule Instantiation Procedure)**.** Let $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ be an arbitrary partition. Suppose $l \to r \in \mathcal{R}_c$ and there exists $\sigma = \mathrm{mgu}(l,r)$ such that σ is a linear substitution. Put $\mathcal{R}_2 = (\mathcal{R} \setminus \{l \to r\}) \cup \{l\rho \to r\rho \mid \rho \in \mathrm{Comp}(\sigma)\}$.

▶ **Theorem 12.** *Suppose that $\mathcal{R}^{\succ}$ is strongly quasi-reducible and $\mathcal{R}_2$ is obtained from $\mathcal{R}$ by the rule instantiation procedure. If $\mathcal{R}_2$ is ground confluent, then so is $\mathcal{R}$.*

Proof. Suppose $\mathcal{R}^{\succ}$ is strongly quasi-reducible, $l \to r \in \mathcal{R}$ and $\sigma = \mathrm{mgu}(l,r)$ is a linear substitution. Let $\mathcal{S} = \{l\delta \to r\delta \mid \delta \in \mathrm{Comp}(\sigma)\}$ and $\mathcal{S}' = \mathcal{S} \cup \{l\sigma \to r\sigma\}$. Let $\mathcal{R}' = \mathcal{R} \cup \mathcal{S}$. Then, since $l \to r \in \mathcal{R}$, we have $\to_{\mathcal{R}'} = \to_{\mathcal{R}}$, and thus $\mathcal{R}$ is ground confluent iff so is $\mathcal{R}'$. We now show if $\mathcal{R}_2$ is ground confluent then $\mathcal{R}'$ is ground confluent. To show this, from $\mathcal{R}' = \mathcal{R}_2 \cup \{l \to r\}$ and Proposition 2, it suffices to show $\mathcal{R}_2 \models_{ind} l \doteq r$. Since l and r are unifiable and $\succ$ is well-founded, we have $l \to r \notin \mathcal{R}^{\succ}$. Thus $\mathcal{R}^{\succ} \subseteq \mathcal{R}_2$. Hence, as $\mathcal{R}^{\succ}$ is strongly quasi-reducible and terminating, for any ground substitution θ_g there exists a ground constructor substitution θ_{gc} such that $\theta_g(x) \to^*_{\mathcal{R}_2} \theta_{gc}(x)$. Thus for any ground substitution θ_g, we have $l\theta_g \to^*_{\mathcal{R}_2} l\theta_{gc}$ and $r\theta_g \to^*_{\mathcal{R}_2} r\theta_{gc}$. By the property of complement, mentioned above, for any $l\theta_{gc}$, there exists $\delta \in \mathrm{Comp}(\sigma) \cup \{\sigma\}$ such that $l\theta_{gc}$ is an instance of $l\delta$. Thus, $l\theta_{gc} \to_{\mathcal{S}'} r\theta_{gc}$. Since $l\sigma = r\sigma$, we have $l\theta_{gc} \to^{=}_{\mathcal{S}} r\theta_{gc}$. As $\mathcal{S} \subseteq \mathcal{R}_2$, we have $l\theta_g \leftrightarrow^*_{\mathcal{R}_2} r\theta_g$. Hence $\mathcal{R}_2 \models_{ind} l \doteq r$. ◀

4 Relaxing Ordering-Constraints in Rewriting Induction

The technique in the previous section enables us to deal with non-orientable rule $l \to r \in \mathcal{R}$ if it can be moved to conjecture part and replaced with strictly decreasing rule. However, if the rule is a constructor rule, this may be not possible in nature.

▶ **Example 13.** Let $\mathcal{F} = \{\mathsf{sum} : \mathsf{Btree} \to \mathsf{Nat},\ + : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat},\ \mathsf{node} : \mathsf{Nat} \times \mathsf{Btree} \times \mathsf{Btree} \to \mathsf{Btree},\ \mathsf{leaf} : \mathsf{Nat} \to \mathsf{Btree},\ \mathsf{s} : \mathsf{Nat} \to \mathsf{Nat},\ \mathsf{0} : \mathsf{Nat}\}$ and $\mathcal{R}$ be the following TRS:

$$\left\{ \begin{array}{llllll} \mathsf{sum}(\mathsf{leaf}(x)) & \to & x & \mathsf{sum}(\mathsf{node}(x,yt,zt)) & \to & +(x,+(\mathsf{sum}(yt),\mathsf{sum}(zt))) \\ +(x,\mathsf{0}) & \to & x & +(x,\mathsf{s}(y)) & \to & \mathsf{s}(+(x,y)) \\ \mathsf{node}(x,yt,zt) & \to & \mathsf{node}(x,zt,yt) & & & \end{array} \right\}$$

Modify

$$\frac{\langle E \uplus \{s \doteq t\},\ H\rangle}{\langle E \cup \{s' \doteq t\},\ H\rangle} \quad s \overset{\approx}{\to}_{\mathcal{R}} s'$$

Figure 2 *Modify* inference rule.

Now the non-orientable $\mathsf{node}(x, yt, zt) \to \mathsf{node}(x, zt, yt)$ can not be replaced with strictly decreasing rules nor moved to the conjecture part. Thus, the previous techniques cannot deal with this case.

To deal with such a case, we first extend our rewriting induction system. The idea is to allow a constructor rewrite rule $l \to r$ such that $l \approx r$. We assume that a TRS $\mathcal{R}$ and a reduction quasi-order $\succsim$ satisfy $\mathcal{R} \subseteq \succsim$ and $\mathcal{R}^{\succ}$ is strongly quasi-reducible.

▶ **Definition 14.** Inference rules of rewriting induction for proving bounded ground convertibility of many-sorted TRSs is obtained from those given in Figure 1 by (i) adding *Modify* rule in Figure 2, and (ii) replacing Expd operation in *Expand* rule with $\mathrm{Expd}^{\succ}$ defined as follows:

$$\mathrm{Expd}^{\succ}_{u}(s,t) \quad = \quad \{C[r]\mu \to t\mu \mid \mu = \mathrm{mgu}(l,u), l \succ r, l \to r \in (\mathcal{R} \setminus \mathcal{R}_c) \cup (\mathcal{R}_c \otimes f)\},$$

where $s = C[u]$ and $u(\epsilon) = f$.

We now provide the key property of the system (see Appendix B for the proof).

▶ **Theorem 15.** *If $\langle E, \emptyset\rangle \rightsquigarrow^* \langle \emptyset, H\rangle$ for some H, then E is bounded ground convertible by $\mathcal{R}$ w.r.t. $\succsim$.*

Allowing (constructor) rewrite rules $l \to r$ such that $l \approx r$ makes the bounded ground convertibility alone insufficient for guaranteeing ground confluence. Thus, we need an extension of Newman's Lemma [15] to suit our situation (see Appendix A for the proof).

▶ **Lemma 16.** *Let $\succsim$ be a well-founded quasi-order. Suppose $\mathcal{R}_d = \mathcal{R} \setminus \mathcal{R}_c$, $\mathcal{R}_d \subseteq \succ$, $\mathcal{R}_c \subseteq \succsim$ and $\mathcal{R}_c$ is ground confluent. Then, $\mathcal{R}$ is ground confluent if the following two conditions are satisfied for any ground terms u_g, v_g:*

(i) *$u_g \overset{\prec}{\leftarrow}_{\mathcal{R}} \circ \to_{\mathcal{R}_d} v_g$ implies $u_g \leftrightarrow^*_{\mathcal{R}^{\prec}_{\succsim}} v_g$, and*

(ii) *$u_g \overset{\approx}{\leftarrow}_{\mathcal{R}} \circ \to_{\mathcal{R}_d} v_g$ implies $u_g \to_{\mathcal{R}_d} \circ \leftrightarrow^*_{\mathcal{R}^{\prec}_{\succsim}} v_g$.*

Henceforth, we put $\mathcal{R}_d = \mathcal{R} \setminus \mathcal{R}_c$ and assume $\mathcal{R}_d \subseteq \succ$, $\mathcal{R}_c \subseteq \succsim$ (thus, $\mathcal{R}^{\approx} \subseteq \mathcal{R}_c$) and $\mathcal{R}^{\succ}$ is strongly quasi-reducible. The idea to obtain a ground confluence proof of $\mathcal{R}$ by our new rewriting induction system for bounded ground convertibility is to apply Lemma 16 with the help of the assumption that $\mathcal{R}_c$ is ground confluent and Theorem 15. The condition (i) of the lemma is guaranteed by requesting $\langle \mathrm{CP}(\mathcal{R}^{\succ}) \setminus \mathrm{CP}(\mathcal{R}_c^{\succ}), \emptyset\rangle \rightsquigarrow^* \langle \emptyset, H\rangle$, thanks to Theorem 15.

To guarantee the condition (ii) of the lemma, we assume $\mathcal{R}$ is supplemented by $\{l \to r \mid l \succ r, l \to r \in \mathcal{R}_c \otimes f, f \in \mathcal{D}\}$ so that $\mathcal{R}_d$ is quasi-reducible. This is possible since we assume that $\mathcal{R}^{\succ}$ is strongly quasi-reducible, and the addition does not affect whether $\mathcal{R}$ is ground confluent or not. Furthermore, we need to modify inference rules of rewriting induction and the starting set of equations from $\mathrm{CP}(\mathcal{R}_c^{\approx}, \mathcal{R}_d)$ or $\mathrm{CP}(\mathcal{R}_d, \mathcal{R}_c^{\approx})$ by marking the left-hand side

Expand

$$\frac{\langle E \uplus \{s^\circ \doteq t\},\ H\rangle}{\langle E \cup \{s'_i \doteq t_i\}_i,\ H \cup \{s \to t\}\rangle} \quad \begin{array}{l} u \in \mathcal{B}(s), \{s_i \to t_i\}_i = \mathrm{Expd}^{\succ}_u(s,t), \\ s_i \overset{\succsim}{\to}{}^*_{H\leftrightarrow} s'_i \end{array}$$

Simplify

$$\frac{\langle E \uplus \{s^\circ \doteq t\},\ H\rangle}{\langle E \cup \{s' \doteq t\},\ H\rangle} \quad s \overset{\succ}{\to}_{\mathcal{R}^\circ \cup H} \circ \overset{\succsim}{\to}{}^*_{H\leftrightarrow} s'$$

Modify

$$\frac{\langle E \uplus \{s \doteq t\},\ H\rangle}{\langle E \cup \{s' \doteq t\},\ H\rangle} \quad s \overset{\approx}{\to}_{\mathcal{R}} s'$$

Delete

$$\frac{\langle E \uplus \{s^\circ \doteq t\},\ H\rangle}{\langle E,\ H\rangle} \quad s \to^{=}_{H} t$$

Figure 3 Inference rules of rewriting induction for ground confluence proof.

or the right-hand side which is required to apply $\mathcal{R}_d$. We denote a marked term s as $s^\bullet$ and a marked equation as $s^\bullet \doteq t$ or $s \doteq t^\bullet$. Then we have the following set of equations.

$$\begin{aligned} \mathrm{CP}_{\gtrsim}(\mathcal{R}) \quad = \quad & \mathrm{CP}(\mathcal{R}^{\succ}) \setminus \mathrm{CP}(\mathcal{R}^{\succ}_c) \cup \{s^\bullet \doteq t \mid s \mathrel{\overset{\frown}{\cup}} t \in \mathrm{CP}(\mathcal{R}^{\approx}_c, \mathcal{R}_d)\} \\ & \cup \{s \doteq t^\bullet \mid s \mathrel{\overset{\frown}{\cup}} t \in \mathrm{CP}(\mathcal{R}_d, \mathcal{R}^{\approx}_c)\} \end{aligned}$$

By extending *Expand*, *Simplify* and *Delete* rules of rewriting induction for treating marked equations, we obtain inference rules of Figure 3 for proving ground confluence of $\mathcal{R}$.

▶ **Definition 17.** Inference rules of rewriting induction for proving ground confluence of many-sorted TRSs is given in Figure 3. Here, E is the set of marked or unmarked equations, and s° denotes a marked term $s^\bullet$ or an unmarked term s. In *Simplify* rule, $\mathcal{R}^\circ$ denotes $\mathcal{R}_d$ if s° is a marked term or $\mathcal{R}$ if s° is an unmarked term.

Note that only *Expand*, *Simplify* or *Delete* rule is applied to a marked equation, and then, the mark is removed; in other words, marked equations can be removed only by applying one of them. For unmarked equations, any of four inference rules is applied.

Using our new rewriting induction system for ground confluence proof, we obtain the following new ground confluence criterion (see Appendix B for the proof).

▶ **Theorem 18.** *Let $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ be an arbitrary partition. Suppose that $\gtrsim$ is a reduction quasi-order such that $\mathcal{R} \subseteq \gtrsim$. Suppose that $\mathcal{R}^{\succ}$ is left-linear and strongly quasi-reducible, $\mathcal{R}^{\approx} \subseteq \mathcal{R}_c$ and $\mathcal{R}_c$ is ground confluent. If $\langle \mathrm{CP}_{\gtrsim}(\mathcal{R}), \emptyset\rangle \overset{*}{\rightsquigarrow} \langle \emptyset, H\rangle$ for some H, then $\mathcal{R}$ is ground confluent.*

▶ **Example 19.** Let $\mathcal{R}$ be the TRS given in Example 13. Take $\mathcal{C} = \{0, \mathsf{s}, \mathsf{leaf}, \mathsf{node}\}$, $\mathcal{R}_c = \mathcal{R}^{\approx} = \{\mathsf{node}(x, yt, zt) \to \mathsf{node}(x, zt, yt)\}$ and $\mathcal{R}_d = \mathcal{R}^{\succ}$. It is obvious that $\mathcal{R}^{\succ}$ is left-linear and strongly quasi-reducible and $\mathcal{R}_c$ is ground confluent. Then we have $\mathrm{CP}_{\gtrsim}(\mathcal{R}) = \{\mathsf{sum}(\mathsf{node}(x, zt, yt))^\bullet \doteq +(x, +(\mathsf{sum}(yt), \mathsf{sum}(zt)))\}$. Applying inference rules of Figure 3, we obtain $\langle \mathrm{CP}_{\gtrsim}(\mathcal{R}), \emptyset\rangle \overset{*}{\rightsquigarrow} \langle \emptyset, H\rangle$, where

$$H = \left\{ \begin{array}{lcl} +(x, +(\mathsf{s}(y), z)) & \to & \mathsf{s}(+(x, +(z, y))), \\ +(x, +(0, y)) & \to & +(x, y), \\ +(x, +(y, z)) & \to & +(x, +(z, y)), \\ +(x, +(y, \mathsf{sum}(zt))) & \to & +(x, +(\mathsf{sum}(zt), y)), \\ +(x, +(\mathsf{sum}(yt), \mathsf{sum}(zt))) & \to & +(x, +(\mathsf{sum}(zt), \mathsf{sum}(yt))) \end{array} \right\}.$$

Thus, from Theorem 18 it follows that $\mathcal{R}$ is ground confluent.

5 Disproving Ground Confluence

In this section, we deal with methods to disprove ground confluence. We present two mechanisms to disprove ground confluence – the first one is given as an additional inference rule of rewriting induction and the second one is a general method that is obtained by modifying a method for disproving confluence [1]. From here on, for some set A of terms, rules, etc. we denote those of sort τ by A^τ if the sort information is necessary.

Non-confluence is usually shown by selecting some candidates of non-joinable term-pair $\langle t, u\rangle$ such that $t \,{}^*\!\leftarrow \circ \to^* u$, and proving t and u are not joinable. This idea is naturally incorporated into the setting of ground confluence disproving. However, one needs to be careful about the rewrite sequence for counterexample can be instantiated by ground terms; that is, it may happen that $t_g \,{}^*\!\leftarrow s \to^* u_g$ for ground terms u_g, t_g and term s, but there is no ground instantiation of s, and in such a case, even if t_g and u_g are not joinable, this does not imply ground non-confluence of the system. To avoid such pitfalls, it is better to exclude rewrite rules that can not be instantiated by ground terms. This motivates us to introduce the following definitions.

▶ **Definition 20** (redundant rewrite rules)**.** A sort τ is said to be *redundant* if $\mathcal{T}(\mathcal{F})^\tau = \emptyset$. A rule $l \to r \in \mathcal{R}$ is redundant if there exists a redundant sort τ such that (1) $l \to r \in \mathcal{R}^\tau$ or (2) $\mathcal{V}(l) \cap \mathcal{V}^\tau \neq \emptyset$. A TRS $\mathcal{R}$ is non-redundant if $\mathcal{R}$ contains no redundant rules.

It is easy to see redundancy is decidable. Since removing redundant rules preserves ground (non-)confluence, one can work with non-redundant TRSs to prove or disprove ground confluence:

▶ **Theorem 21.** *Let $\mathcal{R}'$ be obtained by removing redundant rules in $\mathcal{R}$. Then, $\mathcal{R}'$ is ground confluent if and only if so is $\mathcal{R}$.*

Proof. It suffices to have $\to_{\mathcal{R}} = \to_{\mathcal{R}'}$ on $\mathrm{T}(\mathcal{F})$. ($\supseteq$) is clear as $\mathcal{R}' \subseteq \mathcal{R}$. Suppose $s_g \to_{\mathcal{R}} t_g$. Then $s_g \to_{l\to r} t_g$ for some non-redundant rule $l \to r \in \mathcal{R}$ since no redundant rule is applied in a ground rewrite step. Thus, $s_g \to_{\mathcal{R}'} t_g$. ◀

Any rewrite sequence $s \to^* t$ by non-redundant rules has a ground instantiation. Thus, for non-redundant TRS $\mathcal{R}$, if $t \,{}^+_{\mathcal{R}}\!\leftarrow \circ \to^+_{\mathcal{R}} u$ and there exists ground instantiations $t\theta_g$ and $u\theta_g$ which are not joinable, then $\mathcal{R}$ is not ground confluent. *In the remainder of this section, we assume $\mathcal{R}$ is non-redundant.*

Several inference rules of rewriting induction for refuting inductive conjectures are known (e.g. [6, 14]) – for example, when $\mathcal{R}$ has free constructors, if there exists $s \doteq t \in E$ with $s(\epsilon), t(\epsilon) \in \mathcal{C}$ and $s(\epsilon) \neq t(\epsilon)$, one can infer $\bot$ from $\langle E, H\rangle$, meaning that the initial equation of the derivation is not an inductive theorem of $\mathcal{R}$. Here, the correctness of such rules is guaranteed by the assumption that $\mathcal{R}$ *is confluent* [14]. On the other hand, since we are dealing with proving ground confluence of $\mathcal{R}$, it is not appropriate to assume *confluence* of $\mathcal{R}$. However, as we will see below, it turns out that not the same but a similar idea can be used for refuting ground confluence.

First, we need a couple of preparations before presenting our inference rule for refuting ground confluence.

▶ **Definition 22.** A function symbol f is said to be *stable* (in a TRS $\mathcal{R}$) if for any rewrite rule $l \to r \in \mathcal{R}$, $f = l(\epsilon)$ implies $f = r(\epsilon)$. A term s is *root-stable* if $s(\epsilon) \in \mathcal{F}$ and $s(\epsilon)$ is stable.

Disprove

$$\frac{\langle E \cup \{s \doteq t\},\ H\rangle}{\bot} \quad s \not\approx_{\mathcal{R}} t$$

Figure 4 Inference rule for disproving.

A term s is a *minimal form* (of a TRS $\mathcal{R}$) if $s \to_{\mathcal{R}} t$ implies $s = t$. A term is a ground minimal form if it is ground and a minimal form. We denote the set of ground minimal forms of $\mathcal{R}$ of sort τ by $\mathrm{GMF}(\mathcal{R})^\tau$.

We are now ready to present the key definition of our inference rule.

▶ **Definition 23.** For terms s and t of the same sort, we write $s \not\approx_{\mathcal{R}} t$ if either

(i) s, t are root-stable (in $\mathcal{R}$) with $s(\epsilon) \neq t(\epsilon)$,
(ii) $s(\epsilon) = t(\epsilon) \notin \{l(\epsilon) \mid l \to r \in \mathcal{R}\}$ and $s|_i \not\approx t|_i$ for some $1 \leq i \leq \mathrm{arity}(s(\epsilon))$,
(iii) $s \in \mathcal{V}^\tau \setminus \mathcal{V}(t)$ and there exist $u_g, v_g \in \mathrm{GMF}(\mathcal{R})^\tau$ such that $u_g \neq v_g$, or
(iv) $s \in \mathcal{V}^\tau$, t is root-stable and there exists a root-stable term u of sort τ such that $u(\epsilon) \neq t(\epsilon)$.

The intended property of the relation $\not\approx_{\mathcal{R}}$ is as follows (see Appendix C for the proof).

▶ **Lemma 24.** *Suppose $\mathcal{V}(s) \cup \mathcal{V}(t)$ contains no variable of redundant sort. If $s \not\approx_{\mathcal{R}} t$, then there exists a ground substitution θ_g such that $s\theta_g$ and $t\theta_g$ are not joinable by $\mathcal{R}$.*

▶ **Definition 25.** Rewriting induction with disproof is obtained by adding the inference rule in Figure 4, where $\mathcal{R}$ is the input TRS of the problem.

▶ **Lemma 26.** *Let $\langle E_0, \emptyset\rangle \overset{*}{\rightsquigarrow} \langle E_i, H_i\rangle$. (1) $l \leftrightarrow^*_{E_0 \cup \mathcal{R}} r$ for any $l \doteq r \in E_i$ and $l \to r \in H_i$. (2) $\mathcal{R} \models_{ind} E_0$ then $\mathcal{R} \models_{ind} E_i$.*

Proof. (1) Straightforward, using induction on the length of $\langle E_0, \emptyset\rangle \overset{*}{\rightsquigarrow} \langle E_i, H_i\rangle$. (2) Immediately follows from (1). ◀

▶ **Theorem 27.** *Suppose $\mathcal{R}$ is non-redundant and $\mathcal{R} \models_{ind} \mathcal{R}_0$. If $\langle \mathrm{CP}(\mathcal{R}_0) \cup (\mathcal{R} \backslash \mathcal{R}_0), \emptyset\rangle \overset{*}{\rightsquigarrow} \bot$ then $\mathcal{R}$ is not ground confluent.*

Proof. Suppose $\langle \mathrm{CP}(\mathcal{R}_0) \cup (\mathcal{R} \backslash \mathcal{R}_0), \emptyset\rangle \overset{*}{\rightsquigarrow} \langle E, H\rangle \rightsquigarrow \bot$. Then, by the definition of *Disprove* inference rule, there exists $s \doteq t \in E$ such that $s \not\approx_{\mathcal{R}} t$. Hence, by Lemma 24, there exists ground instances $s\theta_g, t\theta_g$ that are not joinable by $\mathcal{R}$. On the other hand, since $\mathcal{R} \models_{ind} \mathrm{CP}(\mathcal{R}_0) \cup (\mathcal{R} \backslash \mathcal{R}_0)$, we have $\mathcal{R} \models_{ind} s \doteq t$ by Lemma 26. Hence, by non-redundancy of $\mathcal{R}$, we have a ground rewrite sequence $s\theta_g \leftrightarrow^*_{\mathcal{R}} t\theta_g$. Thus, $\mathcal{R}$ is not ground confluent. ◀

Note here the Disprove inference rule should be used with the side condition $s \not\approx_{\mathcal{R}} t$ and not with $s \not\approx_{\mathcal{R}_0} t$. This is a sharp contrast with other inference rules, which are concerned with $\mathcal{R}_0$ (and not with $\mathcal{R}$). The following example illustrates such replacement is incorrect.

▶ **Example 28.** Let $\mathcal{R}$ be as follows:

$$\{\ \mathsf{b} \to \mathsf{c}\ \ (a) \qquad \mathsf{c} \to \mathsf{b}\ \ (b) \qquad \mathsf{b} \to \mathsf{a}\ \ (c)\ \}$$

Note that $\mathcal{R}$ is ground confluent. Take $\mathcal{C} = \{\mathsf{a}, \mathsf{c}\}$ and $\mathcal{R}_0 = \{(a)\}$. Then, $\mathrm{CP}(\mathcal{R}_0) \cup (\mathcal{R} \backslash \mathcal{R}_0) = \{(b), (c)\}$, and thus, a rewriting induction derivation $\langle \{(b), (c)\}, \emptyset\rangle \rightsquigarrow^* \langle \{\mathsf{c} \doteq \mathsf{a}\}, \emptyset\rangle$. Now, c, a are root-stable in $\mathcal{R}_0$ and $\mathsf{c} \neq \mathsf{a}$. Hence, $\mathsf{c} \not\approx_{\mathcal{R}_0} \mathsf{a}$. Thus, if we had replaced the condition $s \not\approx_{\mathcal{R}} t$ with $s \not\approx_{\mathcal{R}_0} t$, we could have derived $\bot$.

In [5] a procedure with SPIKE for disproving ground confluence is proposed, but it works on the assumption that $\mathcal{R}$ is terminating; thus, it cannot deal with non-orientable constructor rules. On the other hand, Theorem 27 can be applied to $\mathcal{R}$ having non-orientable constructor rules.

To end the section, we explain how methods for disproving confluence [1] can be incorporated to prove ground non-confluence. The basic idea of disproving confluence is to take some terms s, t such that $s \ {}^{+}\!\leftarrow \circ \rightarrow^{+} t$ and show s and t are not joinable. Since such s, t are obtained by taking forward closure or backward closure of rewrite rules and computing critical pairs.

In the case of disproving ground confluence we first instantiate s, t with some ground terms. Thus, we take some $\langle s\theta_g, t\theta_g \rangle$ as a candidate. The rest of technique to show non-joinability of $s\theta_g, t\theta_g$ remain the same. Note we first need to remove redundant rules beforehand, otherwise, $s \ {}^{+}\!\leftarrow \circ \rightarrow^{+} t$ may not have ground instantiation, and the non-joinability of $s\theta_g, t\theta_g$ does not necessarily implies ground non-confluence.

6 Implementation and Experiments

All the techniques presented in this paper have been implemented in our ground confluence prover AGCP [2]. Our improved procedure is described as follows.

Improved GCR Procedure

Input: many-sorted TRS $\mathcal{R}$
Output: `YES`, `NO` or `MAYBE`

1. Check redundancy of sorts and remove redundant rules (Section 5).
2. Compute (possibly multiple) candidates for the partition $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ of function symbols.
3. Select constructor rules from $\mathcal{R}$ and construct a constructor subsystem $\mathcal{R}_c$ by the rule instantiation procedure (Section 3).
4. For each $f \in \mathcal{D}$, construct multiple candidates of defining rules for f using the rule complementation procedure (Section 3), and construct $\mathcal{R}_0$ from $\mathcal{R}_c$ by adding a candidate for each.
5. Find a reduction quasi-order $\succsim$ satisfying conditions of Theorem 18 for $\mathcal{R}_0$ (Section 4). If it fails try another candidate of defining rules; if the candidates are exhausted, try another partition $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$. Run rewriting induction to obtain $\langle \mathrm{CP}_{\succsim}(\mathcal{R}_0) \cup (\mathcal{R} \setminus \mathcal{R}_0), \emptyset \rangle \overset{*}{\rightsquigarrow} \langle \emptyset, H \rangle$ for some H or $\langle \mathrm{CP}_{\succsim}(\mathcal{R}_0) \cup (\mathcal{R} \setminus \mathcal{R}_0), \emptyset \rangle \overset{*}{\rightsquigarrow} \bot$. (Sections 4 and 5). If it succeeds, return `YES` or `NO` accordingly. If the number of rewriting induction steps exceeds a limit, then try another candidate of defining rules.
6. Run the ground non-confluence check incorporated from the methods for disproving confluence [1] (Section 5). If it fails, return `MAYBE`.

Below we explain some details of the procedure and heuristics employed in our implementation.

- We take $l \rightarrow r \in \mathcal{R}$ satisfying $\mathcal{F}(l) \cup \mathcal{F}(r) \subseteq \mathcal{C}$ as a constructor rule; those $\mathcal{F}(l) \cup \mathcal{F}(r) \not\subseteq \mathcal{C}$ and $l(\epsilon) \notin \mathcal{D}$ are moved to the conjecture part. Succeedingly, $\mathcal{R}_c$ is modified by the rule instantiation procedure in Section 3. For checking ground confluence of $\mathcal{R}_c$, few basic techniques for checking confluence are employed.

Table 1 Test on 244 examples of CoCo 2016 GCR demonstration category.

	AGCP (A) (coco2016)	FORT (F) (coco2016)	ALL	$A \setminus ALL$	$F \setminus ALL$
YES	105	83	140	0	43
NO	0	32	34	0	2
YES+NO	105	115	174	0	45

- For each $f \in \mathcal{D}$, we construct multiple candidates of defining rules for f as follows. Firstly, from $\mathcal{R}$, we exclude the rules whose lhs is not a linear basic term and those having a subterm of r that unifies with l, and such rules are moved to the conjecture part; let the result be $\mathcal{S}$. Then, we try to select minimal subsets of f-rules in $\mathcal{S}$ that is strongly quasi-reducible. If there is no such subset, we perform three ways to supplement rewrite rules using the rule complementation procedure in Section 3.
  - If $\mathrm{LHS}(f, \mathcal{S})$ is non-empty, a complement pattern of $\mathrm{LHS}(f, \mathcal{S})$ is computed and rewrite rules for the such patterns are added.
  - If $\mathrm{LHS}(f, \mathcal{S})$ is empty, we take basic patterns such as only i-th argument is extended (i.e. $\{f(x_1, \ldots, x_{i-1}, u, x_{i+1}, \ldots, x_n) \mid u = c(\bar{y}), c \in \mathcal{C}\}$) and those and all arguments are extended, and rewrite rules for the such patterns are added.
  - If there is a rule such that recursive call on the rhs of the rule has an argument, say u, whose root symbol is a defined symbol, then we seek for rewrite rules $l \to r$ and σ such that $\sigma = \mathrm{mgu}(l, u)$ and take a rewrite rule for the pattern $l\sigma$.

  Each of these cases, for each pattern l, we seek for reducts of l within some steps, and take a term r of minimal size to generate an added rule $l \to r$.
- A reduction quasi-order used in the rewriting induction is selected from recursive path orders with multiset and lexicographic status. We obtain it by encoding ordering constraints expressing that there is at least one candidate of strictly decreasing strongly quasi-reducible defining rules for each defined function symbol and constructor rules are weakly decreasing, and solving them using an SMT-solver.
- For the marked term $t^\bullet$, we try to remove the mark at the very first steps of the rewriting induction.
- *Modify* rules is used before applying *Simplify* or *Delete* rules – this is a similar heuristics to the one how weakly decreasing rewrite steps by $H^{\leftrightarrow}$ are included [2].
- We impose a limitation of the length of rewriting induction derivation; when the length reaches the limit, we seek for a defined symbol, say f, whose defining rules may be a cause of the divergence by checking the root of innermost basic subterms of rewrite rules in H-part. We then switch the defining rules for f to the next candidate.

We have tested our prover on the collection of 244 examples that was used in the GCR demonstration category of Confluence Competition 2016. The collection is contributed by two participants AGCP [2] and FORT [12] of the category. The result is summarized in Table 1. Our new prover succeeds in proving ground confluence or non-confluence for 174 problems in total (shown in the column below ALL), where as AGCP and FORT prove 105 and 115, respectively (shown in the columns below AGCP and FORT). The row titled YES shows the number of success in ground confluence proving and the one titled NO shows the number of success in ground non-confluence proving. The total increase from AGCP is 65, and the increase in proving ground confluence is 35 and those in proving ground non-confluence is 34. $A \setminus ALL$ ($F \setminus ALL$) denotes the number of problems for which A (resp. F) succeeds but our prover fails. Table 2 shows the analysis on which technique contributes the success in 69

Table 2 Analyzing effective techniques in 69 examples from $ALL \setminus A$

	redun-ancy	RI disproof	general disproof	comple-mentation	instan-tiation	non-ori. con. rules	others	total
YES	2	–	–	13	13	4	6	35
NO	(34)	6	28	–	–	-	0	34

Table 3 Test on new 55 examples

	ACP	AGCP	ALL
YES	0	13	30
NO	43	0	3
MAYBE	12	42	22

examples in $ALL \setminus A$. It can be seen that each of presented techniques affects for increasing the power of the prover. "Other" refers to improvements using more sophisticated strategies and reduction orderings.

Many problems in the collection of 244 examples of CoCo 2016 GCR demonstration category stem from problems for confluence of first-order TRSs, and thus it contains many problems for which confluence of unsorted versions can be proved by the state-of-the-art confluence provers. This motivates us to construct problems which are non-confluent or for which confluence proof can not be handled by the state-of-the-art confluence provers. Thus, we constructed new 55 examples including Examples 3, 4, 9, 13, all of which are non-confluent (indicated by NO) or can not be handled (indicated by MAYBE) by the confluent prover ACP [3]. The result is summarized in Table 3. All the details of the experiments are available at `http://www.nue.ie.niigata-u.ac.jp/tools/agcp/experiments/fscd17/`.

7 Conclusion

In this paper, we have presented methods that strengthen the rewriting induction approach for proving ground confluence of many-sorted term rewriting systems. The first method is concerned with how to replace or supplement initial rewrite rules to obtain an appropriate set of rewrite rules for rewriting induction work. The second method is concerned with the non-orientable constructor rules for which the first method can not deal with. For this, we have extended rewriting induction inference system to deal with weakly decreasing rewrite rules. Then we obtain a correctness criteria for the ground confluence proving so that one can deal with non-orientable constructor rules. We believe that our extension provides a basis for dealing with term rewriting systems acting on more flexible data structures. As the last ingredient, we have presented methods to deal with proving ground non-confluence. All of these techniques have been implemented and experiments have shown that presented methods are effective to deal with problems for which state-of-the-art ground confluence provers can not handle.

Acknowledgements. Thanks are due to the anonymous reviewers.

References

1 T. Aoto. Disproving confluence of term rewriting systems by interpretation and ordering. In *Proc. of 9th FroCoS*, volume 8152 of *LNAI*, pages 311–326. Springer-Verlag, 2013.

2 T. Aoto and Y. Toyama. Ground confluence prover based on rewriting induction. In *Proc. of 1st FSCD*, volume 52 of *LIPIcs*, pages 33:1–12. Schloss Dagstuhl, 2016. doi:10.4230/LIPIcs.FSCD.2016.33.

3 T. Aoto, Y. Yoshida, and Y. Toyama. Proving confluence of term rewriting systems automatically. In *Proc. of 20th RTA*, volume 5595 of *LNCS*, pages 93–102. Springer-Verlag, 2009.

4 K. Becker. Proving ground confluence and inductive validity in constructor based equational specifications. In *Proc. of 4th TAPSOFT*, volume 668 of *LNCS*, pages 46–60. Springer-Verlag, 1993.

5 A. Bouhoula. Simultaneous checking of completeness and ground conflunce for algebraic specifications. *ACM Transactions on Computational Logic*, 10(2):20:1–33, 2009.

6 A. Bouhoula, E. Kounalis, and M. Rusinowitch. Automated mathematical induction. *Journal of Logic and Computation*, 5(5):631–668, 1995.

7 N. Dershowitz and U. S. Reddy. Deductive and inductive synthesis of equational programs. *Journal of Symbolic Computation*, 15:467–494, 1993.

8 H. Ganzinger. Ground term confluence in parametric conditional equational specifications. In *Proc. of 4th STACS*, volume 247 of *LNCS*, pages 286–298. Springer-Verlag, 1987.

9 R. Göbel. Ground confluence. In *Proc. of 2nd RTA*, volume 256 of *LNCS*, pages 156–167. Springer-Verlag, 1987.

10 D. Kapur, P. Narendran, and H. Zhang. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica*, 24(4):395–415, 1987.

11 A. Lazrek, P. Lescanne, and J. J. Thiel. Tools for proving inductive equalities, relative completeness, and ω-completeness. *Information and Computation*, 84:47–70, 1990.

12 F. Rapp and A. Middeldorp. Automating the first-order theory of rewriting for left-linear right-ground rewrite systems. In *Proc. of 1st FSCD*, volume 52 of *LIPIcs*, pages 36:1—-12. Schloss Dagstuhl, 2016.

13 U. S. Reddy. Term rewriting induction. In *Proc. of CADE-10*, volume 449 of *LNAI*, pages 162–177. Springer-Verlag, 1990.

14 S. Shimazu, T. Aoto, and Y. Toyama. Automated lemma generation for rewriting induction with disproof. *JSSST Computer Software*, 26(2):41–55, 2009. In Japanese.

15 Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.

16 V. van Oostrom. Confluence by decreasing diagrams. *Theoretical Computer Science*, 126(2):259–280, 1994.

A Proof of Lemma 16

We first present a couple of preparations. Elements x and y are *convertible below* z if $x = x_0 \leftrightarrow x_1 \leftrightarrow \cdots \leftrightarrow x_n = y$ for some $x_0, \ldots, x_n$ such that $z \succ x_i$ $(0 \leq i \leq n)$; we write $x \leftrightarrow^*_{\prec z} y$ if x and y are convertible below z.

A labelled relation $\to = \bigcup_{i \in \mathcal{L}} \to_i$ is *locally decreasing* if there exists a well-founded partial order $\succ$ on $\mathcal{L}$ such that ${}_l\!\leftarrow \circ \to_m \subseteq \leftrightarrow^*_{\Upsilon l} \circ \to^{=}_m \circ \leftrightarrow^*_{\Upsilon l,m} \circ \overset{=}{{}_l\!\leftarrow} \circ \leftrightarrow^*_{\Upsilon m}$, where $\to_{\Upsilon l} = \bigcup_{i \prec l} \to_i$ and $\to_{\Upsilon l,m} = \to_{\Upsilon l} \cup \to_{\Upsilon m}$. Any locally decreasing relation is confluent [16].

▶ **Lemma 29.** *Let $\succsim$ be a well-founded quasi-order. Suppose $\to = \to_d \cup \to_e$, $\to_d \subseteq \succ$ and $\to_e \subseteq \succsim$. Then, $\to$ is confluent if the following three conditions are satisfied:*

(a) *$y \overset{\prec}{\leftarrow} x \to_d z$ implies $y \leftrightarrow^*_{\prec x} z$,*

(b) *$y \overset{\approx}{\leftarrow} x \to_d z$ implies $y \to_d \circ \leftrightarrow^*_{\prec x} z$, and*

(c) *$y \leftarrow_e \circ \to_e z$ implies $y \to^{=}_e \circ \overset{=}{\leftarrow}_e z$.*

Proof. Consider the source labeling, i.e. the labeling such that each $x \to y$ is labelled with x. Then, we easily obtain that $\to$ is locally decreasing; thus, it is confluent. ◀

The diamond property (c) in Lemma 29 can be relaxed to the confluence property (c') as follows.

▶ **Lemma 30.** *Let $\succsim$ be a well-founded quasi-order. Suppose $\to = \to_d \cup \to_c$, $\to_d \subseteq \succ$ and $\to_c \subseteq \succsim$. Then, $\to$ is confluent if the following three conditions are satisfied:*
(a') *$y \overset{\prec}{\leftarrow} x \to_d z$ implies $y \leftrightarrow^*_{\prec x} z$,*
(b') *$y \overset{\approx}{\leftarrow} x \to_d z$ implies $y \to_d \circ \leftrightarrow^*_{\prec x} z$, and*

(c') *$\to_c$ is confluent.*

Proof. Take $\to^*_c$ as $\to_e$ and apply Lemma 29. We have $\to^*_c \subseteq \succsim$, and the condition (c) in Lemma 29 is obviously satisfied. We next show (b) in Lemma 29. We show ${}^n\overset{\approx}{\leftarrow}_c x \to_d \subseteq \to_d \circ \leftrightarrow^*_{\prec x}$ ($n \geq 0$) by induction on n. Let $y \; {}^n\overset{\approx}{\leftarrow}_c x' \leftarrow_c x \to_d z$. From (b') we have $y \overset{n}{\leftarrow}_c x' \to_d u \leftrightarrow^*_{\prec x} z$ for some u. From induction hypothesis, it follows that $y \to_d \circ \leftrightarrow^*_{\prec x'} u \leftrightarrow^*_{\prec x} z$. As $x \approx x'$, we obtain $y \to_d \circ \leftrightarrow^*_{\prec x} z$. Thus, (b) has been proved. We now show (a) in Lemma 29. Suppose $y \overset{\prec}{\leftarrow} x \to_d z$. The case $y \; {}_d\!\leftarrow x \to_d z$ is clear. Otherwise, $y \; {}^*\!\leftarrow_c y'' \overset{\prec}{\leftarrow}_c y' \; {}^*\overset{\approx}{\leftarrow}_c x \to_d z$ for some y', y''. By (b), $y \; {}^*\!\leftarrow_c y'' \overset{\prec}{\leftarrow}_c y' \to_d \circ \leftrightarrow^*_{\prec x} z$. From $y' \approx x$, we obtain $y \leftrightarrow^*_{\prec x} z$. Thus, (a) has been proved. Now, applying Lemma 29, we obtain that $\to_d \cup \overset{*}{\to}_c$ is confluent. From $\to \subseteq \to_d \cup \to^*_c \subseteq \overset{*}{\to}$, it follows that $\to$ is confluent. ◀

The convertibility below x conditions (a') and (b') in Lemma 30 can be replaced with the bounded convertibility conditions (a") and (b") as follows.

▶ **Lemma 31.** *Let $\succsim$ be a well-founded quasi-order. Suppose $\to = \to_d \cup \to_c$, $\to_d \subseteq \succ$ and $\to_c \subseteq \succsim$. Then, $\to$ is confluent if the following three conditions are satisfied:*
(a") *$y \overset{\prec}{\leftarrow} \circ \to_d z$ implies $y \leftrightarrow^*_{\succsim} z$,*
(b") *$y \overset{\approx}{\leftarrow} \circ \to_d z$ implies $y \to_d \circ \leftrightarrow^*_{\succsim} z$, and*
(c") *$\to_c$ is confluent.*

Proof. From (a") and (b") it is obvious that (a') and (b') in Lemma 30 are satisfied. ◀

Therefore, Lemma 16 follows from Lemma 31 immediately by taking $\to = \to_{\mathcal{R}}$, $\to_c = \to_{\mathcal{R}_c}$ and $\to_d = \to_{\mathcal{R}_d}$.

B Proofs of Theorems 15 and 18

Proof of Theorem 15. Let $\langle E_0, H_0\rangle \rightsquigarrow \langle E_1, H_1\rangle \rightsquigarrow \cdots \rightsquigarrow \langle E_m, H_m\rangle$ with $H_0 = \emptyset$ and $E_m = \emptyset$. Let $E_\infty = \bigcup_i E_i$. As $H_0 = \emptyset$, it easily follows $H_i \subseteq E_\infty$ from the inference rules of rewriting induction. We prove the theorem by contradiction. We first introduce a well-founded order on equations $s \doteq t \succ s' \doteq t'$ by $\{s,t\} \succ_m \{s',t'\}$, where $\succ_m$ denotes the multiset extension of $\succ$. Suppose that E_0 is not bounded ground convertible. Then there exists a minimal ground equation $s\sigma_g \doteq t\sigma_g$ that is an instance of $s \doteq t \in E_\infty$ and not bounded convertible. From the minimality of $s\sigma_g \doteq t\sigma_g$ and the strong quasi-reducibility of $\mathcal{R}^{\succ}$, σ_g must be a ground constructor substitution on $\mathcal{V}(s) \cup \mathcal{V}(t)$. We prove the following claim.

▶ **Claim.** *If $s \doteq t \in E_k$ then there exists $s' \doteq t \in E_{k+1}$ and $s'\sigma_g \doteq t\sigma_g$ is a minimal ground equation that is not bounded convertible.*

Proof of Claim. If no inference rule is applied to $s \doteq t$ in $\langle E_k, H_k\rangle \rightsquigarrow \langle E_{k+1}, H_{k+1}\rangle$, the claim is obvious. We distinguish four cases by the inference rule applied and show a contradiction except for (*Modify*). Note that $H \subseteq E_\infty$ in the following cases. Thus if $s \leftrightarrow_H t$ for ground terms s, t then there exist $u \doteq v \in E_\infty$ and θ_g such that $s = u\theta_g, t = v\theta_g$.

1. (*Expand*) We have $\langle E \uplus \{s \doteq t\}, H\rangle \rightsquigarrow \langle E \cup \{s'_i \doteq t_i\}_i, \{s \to t\} \cup H\rangle$, where $\mathrm{Expd}^{\succ}_u(s,t) = \{s_i \to t_i\}_i$, $u \in \mathcal{B}(s)$, and $s_i \overset{\succ}{\underset{H\leftrightarrow}{}}{}^{*} s'_i$ for each i. Since σ_g is a ground constructor substitution, we have $s\sigma_g \to_{\mathcal{R}} s_i\theta_g \to_{\mathrm{Expd}^{\succ}_u(s,t)} t\sigma_g$ and $s\sigma_g \succ s_i\theta_g$ for some θ_g and i. Thus, $s\sigma_g \to_{\mathcal{R}} s_i\theta_g \overset{\succ}{\underset{H\leftrightarrow}{}}{}^{*} s'_i\theta_g \leftrightarrow_{E_\infty} t\sigma_g$. Then, for any step $u_g \leftrightarrow_H v_g$ in $s_i\theta_g \overset{\succ}{\underset{H\leftrightarrow}{}}{}^{*} s'_i\theta_g$, we have $s\sigma_g \succ u_g, v_g$, and hence $s\sigma_g \doteq t\sigma_g \succ u_g \doteq v_g$. As $s\sigma_g \succ s'_i\theta_g$, $s\sigma_g \doteq t\sigma_g \succ s'_i\theta_g \doteq t\sigma_g$. From the minimality of $s\sigma_g \doteq t\sigma_g$, it follows that $u_g \doteq v_g$ and $s'_i\theta_g \doteq t\sigma_g$ are bounded convertible. Thus $s\sigma_g \doteq t\sigma_g$ is bounded convertible; this contradicts the choice of $s\sigma_g \doteq t\sigma_g$.
2. (*Simplify*) We have $\langle E \uplus \{s \doteq t\}, H\rangle \rightsquigarrow \langle E \cup \{s' \doteq t\}, H\rangle$ for some E, H, where $s \overset{\succ}{\underset{\mathcal{R}\cup H}{}}{} \hat{s} \overset{\succ}{\underset{H\leftrightarrow}{}}{}^{*} s'$. Then, $s \overset{\succ}{\underset{\mathcal{R}\cup H}{}}{} \hat{s} = s_1 \leftrightarrow_H s_2 \leftrightarrow_H \cdots \leftrightarrow_H s_k = s' \leftrightarrow_{E_\infty} t$ with $s_i \succsim s_{i+1}$ for $i = 1, \ldots, k-1$. We distinguish two cases.
   - **a.** Case $s \overset{\succ}{\underset{\mathcal{R}}{}}{} \hat{s}$. Then $s\sigma_g \succ \hat{s}\sigma_g$ and thus $s\sigma_g \succ s_i\sigma_g$ for $i = 1, \ldots, k$. Hence, we have $s\sigma_g \doteq t\sigma_g \succ s_i\sigma_g \doteq s_{i+1}\sigma_g$ for $i = 1, \ldots, k-1$ and $s\sigma_g \doteq t\sigma_g \succ s'\sigma_g \doteq t\sigma_g$. As $s_i\sigma_g \doteq s_{i+1}\sigma_g$ $(i = 1, \ldots, k-1)$ and $s'\sigma_g \doteq t\sigma_g$ are bounded convertible, $s\sigma_g \doteq t\sigma_g$ is bounded convertible, contradiction.
   - **b.** Case $s \overset{\succ}{\underset{H}{}}{} \hat{s}$. Then $s \leftrightarrow_{E_\infty} \hat{s}$ with $s \succ \hat{s}$ and $s\sigma_g \succ s_i\sigma_g$ for all $i = 1, \ldots, k$. Hence, we have $s\sigma_g \doteq t\sigma_g \succ s_i\sigma_g \doteq s_{i+1}\sigma_g$ for $i = 1, \ldots, k-1$ and $s\sigma_g \doteq t\sigma_g \succ s'\sigma_g \doteq t\sigma_g$. Thus, $s_i\sigma_g \doteq s_{i+1}\sigma_g$ $(i = 1, \ldots, k-1)$ and $s'\sigma_g \doteq t\sigma_g$ are bounded convertible. By $s \overset{\succ}{\underset{H}{}}{} \hat{s}$, there exists $w \to \hat{w} \in H$ such that $s = C[w\theta]$ and $\hat{s} = C[\hat{w}\theta]$ for some context C and substitution θ. If $\theta_g = \sigma_g \circ \theta$ is not a constructor ground substitution on $\mathcal{V}(w) \cup \mathcal{V}(\hat{w})$, then $s\sigma_g = C[w\theta]\sigma_g = C\sigma_g[w\theta_g] \overset{+}{\underset{\mathcal{R}}{}}{} s\rho_g$ (or $t\sigma_g \overset{+}{\underset{\mathcal{R}}{}}{} t\rho_g$) for some ground constructor substitution ρ_g and $s\sigma_g \doteq t\sigma_g \succ s\rho_g \doteq t\rho_g$. As $s\rho_g \doteq t\rho_g$ is bounded convertible, it follows that $s\sigma_g \doteq t\sigma_g$ is bounded convertible, contradiction. Thus, suppose that $\theta_g = \sigma_g \circ \theta$ is a constructor ground substitution. As $w \to \hat{w} \in H$, there exists some $p < k$ such that *Expand* rule is applied to $w \doteq \hat{w} \in E_p$ with $u \in \mathcal{B}(w)$ in $\langle E_p, H_p\rangle \rightsquigarrow \langle E_{p+1}, H_{p+1}\rangle$. Thus, from $H_p \subseteq H$ it follows that $w\theta_g \to_{\mathcal{R}} \circ \overset{\succ}{\underset{H\leftrightarrow}{}}{}^{*} \circ \leftrightarrow_{E_\infty} \hat{w}\theta_g$. Hence, $s\sigma_g = C[w\theta]\sigma_g = C\sigma_g[w\theta_g] \to_{\mathcal{R}} w_g \overset{\succ}{\underset{H\leftrightarrow}{}}{}^{*} w'_g \leftrightarrow_{E_\infty} C\sigma_g[\hat{w}\theta_g] = C[\hat{w}\theta]\sigma_g = \hat{s}\sigma_g$ and $s\sigma_g \succ w_g$. Furthermore, for any step $u_g \leftrightarrow_H v_g$ in $w_g \overset{\succ}{\underset{H\leftrightarrow}{}}{}^{*} w'_g$, we have $s\sigma_g \doteq t\sigma_g \succ u_g \doteq v_g$. As $s_i\sigma_g \doteq s_{i+1}\sigma_g$ $(i = 1, \ldots, k-1)$, $s'\sigma_g \doteq t\sigma_g$ and $u_g \doteq v_g$ are bounded convertible, it follows that $s\sigma_g \doteq t\sigma_g$ is bounded convertible, contradiction.
3. (*Modify*) We have $\langle E \uplus \{s \doteq t\}, H\rangle \rightsquigarrow \langle E \cup \{s' \doteq t\}, H\rangle$ for some E, H, where $s \overset{\approx}{\underset{\mathcal{R}}{}}{} s'$. If $s'\sigma_g \doteq t\sigma_g$ is bounded convertible, it follows that $s\sigma_g \doteq t\sigma_g$ is bounded convertible, contradiction. Hence $s'\sigma_g \doteq t\sigma_g$ is not bounded convertible. As $s \approx s'$, the minimality of $s'\sigma_g \doteq t\sigma_g$ is clear.
4. (*Delete*) We have $\langle E \uplus \{s \doteq t\}, H\rangle \rightsquigarrow \langle E, H\rangle$, where $s = t$ or $s \to_H t$. From the choice of $s\sigma_g \doteq t\sigma_g$, we have $s \neq t$. Let $s \to_H t$ and $w \to \hat{w} \in H$ such that $s = C[w\theta]$, $t = C[\hat{w}\theta]$ for some θ. Then, there exists some $p < k$ such that *Expand* rule is applied to $w \doteq \hat{w} \in E_p$ in $\langle E_p, H_p\rangle \rightsquigarrow \langle E_{p+1}, H_{p+1}\rangle$. Hence, in the same way as the case (*Simplify*)-b, it is shown that $s\sigma_g \doteq t\sigma_g$ is bounded convertible, contradiction.

Therefore, *Claim* holds. ◀

From *Claim*, we easily obtain $E_n \neq \emptyset$ $(k \leq n \leq m)$; this contradicts $E_m = \emptyset$. ◀

Proof of Theorem 18. We show the conditions of Lemma 16 are satisfied. To show the condition (i), suppose $u_g \leftarrow_{\mathcal{R}^\succ} \circ \rightarrow_{\mathcal{R}_d} v_g$. If the redexes contracted in the peak $u_g \leftarrow_{\mathcal{R}^\succ} \circ \rightarrow_{\mathcal{R}_d} v_g$ occur at independent positions or variable overlapping positions then it is trivial. If the peak is critical overlapping, then by $\mathrm{CP}(\mathcal{R}^\succ, \mathcal{R}_d) \subseteq \mathrm{CP}_{\succsim}(\mathcal{R})$ and Theorem 15, we obtain $u_g \overset{*}{\leftrightarrow}_{\mathcal{R}^\succsim} v_g$. It remains to show the condition (ii) of Lemma 16, that is, the claim that if $u_g \leftarrow_{\mathcal{R}^\approx} \circ \rightarrow_{\mathcal{R}_d} v_g$ then $u_g \rightarrow_{\mathcal{R}_d} \circ \overset{*}{\leftrightarrow}_{\mathcal{R}^\precsim} v_g$ for ground terms u_g, v_g. If the redexes contracted in the peak $u_g \leftarrow_{\mathcal{R}^\approx} \circ \rightarrow_{\mathcal{R}_d} v_g$ occur at independent positions, then it is trivial. If the peak is variable overlapping then from the left-linearity of $\mathcal{R}^\succ$ and the variable-preserving property of $\mathcal{R}^\approx$, it obviously follows. Suppose that the peak is critical overlapping and $u_g = C[s\theta_g]$, $v_g = C[t\theta_g]$, $s \mathbin{\widehat{\succ}} t \in \mathrm{CP}(\mathcal{R}^\approx_c, \mathcal{R}_d)$ or $t \mathbin{\widehat{\succ}} s \in \mathrm{CP}(\mathcal{R}_d, \mathcal{R}^\approx_c)$. Then, from $s^\bullet \doteq t \in \mathrm{CP}_{\succsim}(\mathcal{R})$ and $\langle \mathrm{CP}_{\succsim}(\mathcal{R}), \emptyset \rangle \overset{*}{\rightsquigarrow} \langle \emptyset, H \rangle$ it follows that *Expand*, *Simplify* or *Delete* rule is eventually applied to $s^\bullet \doteq t$. Thus, in a similar way to the proof of Theorem 15 we obtain that $s\theta_g \rightarrow_{\mathcal{R}_d} \circ \overset{*}{\leftrightarrow}_{\mathcal{R}^\precsim} t\theta_g$ for each applied inference rule. Note here we have $s \neq t$ by $s \mathbin{\widehat{\succ}} t \in \mathrm{CP}(\mathcal{R}^\approx_c, \mathcal{R}_d)$ or $t \mathbin{\widehat{\succ}} s \in \mathrm{CP}(\mathcal{R}_d, \mathcal{R}^\approx_c)$ as $\mathcal{R}_d \subseteq \succ$, and thus $s^\bullet \rightarrow_H t$ if *Delete* is applied. Therefore, $u_g = C[s\theta_g] \rightarrow_{\mathcal{R}_d} \circ \overset{*}{\leftrightarrow}_{\mathcal{R}^\precsim} C[t\theta_g] = v_g$. ◀

C Proof of Lemma 24

Proof. By induction on $|s|$.

1. Suppose $s\theta_g \rightarrow^* u_g \ {}^*\!\leftarrow t\theta_g$. Since $s, t \notin \mathcal{V}$, $s\theta_g(\epsilon) = s(\epsilon)$ and $t\theta_g(\epsilon) = t(\epsilon)$. By root stability, $s\theta_g(\epsilon) = u_g(\epsilon)$ and $t\theta_g(\epsilon) = u_g(\epsilon)$, and thus $s(\epsilon) = t(\epsilon)$. A contradiction to $s(\epsilon) \neq t(\epsilon)$.
2. Suppose $s(\epsilon) = t(\epsilon) \notin \{l(\epsilon) \mid l \rightarrow r \in \mathcal{R}\}$. Then $s\theta_g \rightarrow^* s'$ implies $s|_i\theta_g \rightarrow^* s'|_i$, and $t\theta_g \rightarrow^* t'$ implies $t|_i\theta_g \rightarrow^* t'|_i$. Thus if $s\theta_g \rightarrow^* u_g \ {}^*\!\leftarrow t\theta_g$ then $s\theta_g|_i \rightarrow^* u_g|_i \ {}^*\!\leftarrow t\theta_g|_i$. Thus $s|_i\theta_g$ and $t|_i\theta_g$ are joinable. On the other hand, by induction hypothesis and our assumption $s|_i \not\approx t|_i$, there exists θ_g such that $s|_i\theta_g$ and $t|_i\theta_g$ are not joinable. Hence, $s\theta_g \rightarrow^* u_g \ {}^*\!\leftarrow t\theta_g$ does not hold.
3. Take an arbitrary ground instantiation $t\theta_g$ of t. If $t\theta_g \rightarrow^* u_g$, then take $\rho_g = \theta_g \uplus \{s \mapsto v_g\}$ and then $s\rho_g = v_g$ and u_g are not joinable; hence, so are $s\rho_g$ and $t\rho_g$ $(= t\theta_g)$. Next, suppose otherwise, i.e. $t\theta_g \not\rightarrow^* u_g$. Then, take $\rho_g = \theta_g \uplus \{s \mapsto u_g\}$ and then $s\rho_g = u_g$ and $t\theta_g$ $(= t\theta_g)$ are not joinable by the assumption.
4. Take any ground instance t_g of t. Since $t_g(\epsilon) = t(\epsilon)$ is stable, for any v_g such that $t_g \rightarrow^* v_g$, we have $v_g(\epsilon) = t(\epsilon)$. Let u_g be a ground instance of root-table term u of sort τ. Then u_g is root-stable and $u_g(\epsilon) = u(\epsilon) \neq t(\epsilon)$ by our assumption. Thus $u_g \rightarrow^* u'_g$ implies $u'_g(\epsilon) = u_g(\epsilon)$. Take $\rho_g = \theta_g \uplus \{s \mapsto u_g\}$. Then $s\rho_g = u_g \rightarrow^* \circ \ {}^*\!\leftarrow t\theta_g$ implies $u_g(\epsilon) = t(\epsilon)$, which is a contradiction. ◀

Böhm Reduction in Infinitary Term Graph Rewriting Systems

Patrick Bahr

IT University of Copenhagen, Copenhagen, Denmark
paba@itu.dk

Abstract

The confluence properties of lambda calculus and orthogonal term rewriting do not generalise to the corresponding infinitary calculi. In order to recover the confluence property in a meaningful way, Kennaway et al. [11, 10] introduced Böhm reduction, which extends the ordinary reduction relation so that 'meaningless terms' can be contracted to a fresh constant $\bot$. In previous work, we have established that Böhm reduction can be instead characterised by a different mode of convergences of transfinite reductions that is based on a partial order structure instead of a metric space.

In this paper, we develop a corresponding theory of Böhm reduction for term graphs. Our main result is that partial order convergence in a term graph rewriting system can be truthfully and faithfully simulated by metric convergence in the Böhm extension of the system. To prove this result we generalise the notion of residuals and projections to the setting of infinitary term graph rewriting. As ancillary results we prove the infinitary strip lemma and the compression property, both for partial order and metric convergence.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems, F.1.1 Models of Computation

Keywords and phrases infinitary rewriting, term graphs, Böhm trees

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.8

1 Introduction

A meaningless term (originally called 'undefined element' by Barendregt [6]) is a term that in some sense does not provide any information because it cannot be suitably distinguished from other meaningless terms. In their seminal work on infinitary lambda calculus, Kennaway et al. [11] recognised that infinitary confluence of the infinitary lambda calculus can be established if all meaningless terms are equated. This idea can be elegantly expressed by extending the reduction relation with rules of the form $t \to \bot$ for all meaningless terms of a certain kind. The resulting reduction was coined Böhm reduction. Later, Kennaway et al. [10] applied this idea to first order term rewriting as well.

In previous work [1, 2], we have introduced an alternative approach to deal with meaningless terms that leaves the original reduction relation intact (i.e. no additional rules of the form $t \to \bot$ are needed) but instead changes the underlying model of convergence. Infinitary rewriting, both in the lambda calculus and first-order term rewriting, originally has been based on a metric space to model convergence of transfinite reductions. We showed that if we change the underlying structure from a metric space to a partial order, the resulting infinitary term rewriting system is – under mild assumptions – equivalent to the metric-based system extended to Böhm reduction [1].

In this paper we seek out to develop a corresponding theory of Böhm reduction for term graph rewriting systems in the sense of Barendregt et al. [7] and use it to compare metric-

and partial order-based notions of convergence in a similar way. To this end, we use the theory of infinitary term graph rewriting that was shown to be sound and complete w.r.t. infinitary term rewriting [3]. In this paper, we investigate this approach to infinitary term graph rewriting further and study confluence and convergence properties as well as the relation between the partial order-based and the metric-based mode of convergence.

The main result of this paper is that either mode of convergence can be simulated by the other one if we add rules of the form $g \to \bot$, where g is a root-active term graph. This construction is analogous to the *Böhm reduction* outlined above, and our findings mirror a corresponding result in infinitary term rewriting [2].

As our main proof method we develop a theory of residuals and projections. Some of the theory becomes considerably simpler – compared to infinitary term rewriting – simply because redexes cannot be duplicated by term graph rewriting but are 'shared' instead. On the other hand, many proofs become more tedious as the application of rewrite rules is more complicated than in term rewriting. The theory is put to use in proving the infinitary strip lemma and the compression property, which form the cornerstones in the proof of the main result.

The remainder of this paper is structured as follows: In Section 2, we introduce basic notions of term graphs. Then, in Section 3, we present our infinitary term graph rewriting calculi including their fundamental properties. In Section 4, we develop the theory of residuals and projections, which we then apply to prove the infinitary strip lemma as well the compression lemma; the full account of this development is given in Appendix A. Finally, in Section 5, we use these results in order to prove the equivalence of partial order convergence and metric convergence modulo Böhm extensions as described above. Many proofs in this paper are abridged or have been omitted due to lack of space. The remaining proofs can be found in the extended version of this paper [5].

2 Term Graphs and Modes of Convergence

In this section, we briefly present our notion of term graphs (based on Barendregt et al.[7]) together with the metric and the partial order that are used to formalise infinitary term graph rewriting. For a more thorough exposition, the reader is referred to previous work [3, 4].

Sequences. A *sequence* over a set A of length α is a mapping from an ordinal α into A and is written as $(a_\iota)_{\iota<\alpha}$, which indicates the mapping $\iota \mapsto a_\iota$; the notation $|(a_\iota)_{\iota<\alpha}|$ denotes the length α of the sequence. A sequence is called *open* if its length is a limit ordinal; otherwise it is called *closed*. If $(a_\iota)_{\iota<\alpha}$ is finite it is also written as $\langle a_0, \dots, a_{\alpha-1}\rangle$; in particular, $\langle\rangle$ denotes the empty sequence. A^* denotes the set of finite sequences over A. We write $S \cdot T$ for the *concatenation* of two sequences S and T; S is called a (*proper*) *prefix* of T, denoted $S \leq T$ (resp. $S < T$) if there is a (non-empty) sequence S' such that $S \cdot S' = T$. The uniquely determined prefix of a sequence S of length $\beta < |S|$ is denoted by $S|_\beta$.

Graphs and Term Graphs. A signature Σ is a finite set of symbols together with an associated arity function $\mathsf{ar}(\cdot)$. A *graph* over Σ is a triple $g = (N, \mathsf{lab}, \mathsf{suc})$ consisting of a set N (of *nodes*), a *labelling function* $\mathsf{lab}\colon N \to \Sigma$, and a *successor function* $\mathsf{suc}\colon N \to N^*$ such that $|\mathsf{suc}(n)| = \mathsf{ar}(\mathsf{lab}(n))$ for each node $n \in N$, i.e. a node labelled with a k-ary symbol has precisely k successors. The graph g is called *finite* whenever the underlying set N of nodes is finite. If $\mathsf{suc}(n) = \langle n_0, \dots, n_{k-1}\rangle$, then we write $\mathsf{suc}_i(n)$ for n_i. The successor function suc defines, for each node n, directed edges from n to $\mathsf{suc}_i(n)$. A path from a node m to a node

n in a graph is a finite sequence $\langle e_0, \dots, e_l \rangle$ of numbers such that $n = \mathsf{suc}_{e_l}(\dots \mathsf{suc}_{e_0}(m))$, i.e. n is reached from m by first taking the e_0-th edge, then the e_1-th edge etc.

Given a signature Σ, a *term graph* g over Σ is a tuple $(N, \mathsf{lab}, \mathsf{suc}, r)$ consisting of an *underlying* graph $(N, \mathsf{lab}, \mathsf{suc})$ over Σ whose nodes are all reachable from the *root node* $r \in N$. The term graph g is called *finite* if the underlying graph is finite. The class of all term graphs over Σ is denoted $\mathcal{G}^\infty(\Sigma)$; the class of all finite term graphs over Σ is denoted $\mathcal{G}(\Sigma)$. We use the notation N^g, lab^g, suc^g and r^g to refer to the respective components N,lab, suc and r of g. Given a graph or a term graph h and a node n in h, we write $h|_n$ to denote the sub-term graph of h rooted in n.

Paths, Positions, Term Trees. Let $g \in \mathcal{G}^\infty(\Sigma)$ and $n \in N^g$. A *position* of n is a path in the underlying graph of g from r^g to n. The set of all positions in g is denoted $\mathcal{P}(g)$; the set of all positions of n in g is denoted $\mathcal{P}_g(n)$. The *depth* of n in g, denoted $\mathsf{depth}_g(n)$, is the minimum of the lengths of the positions of n in g, i.e. $\mathsf{depth}_g(n) = \min\{|\pi| \mid \pi \in \mathcal{P}_g(n)\}$. For a position $\pi \in \mathcal{P}(g)$, we write $\mathsf{node}_g(\pi)$ for the unique node $n \in N^g$ with $\pi \in \mathcal{P}_g(n)$, and $g(\pi)$ for $\mathsf{lab}^g(\mathsf{node}_g(\pi))$, i.e. the labelling of g at π. The term graph g is called a *term tree* if each node in g has exactly one position.

Homomorphisms. The notion of homomorphisms is central for dealing with term graphs. For greater flexibility, we will parametrise this notion by a set of constant symbols Δ for which the homomorphism condition is suspended. This will allow us to deal with variables and partiality appropriately. Given $g, h \in \mathcal{G}^\infty(\Sigma)$, a Δ-*homomorphism* ϕ from g to h, denoted $\phi\colon g \to_\Delta h$, is a function $\phi\colon N^g \to N^h$ with $\phi(r^g) = r^h$ that satisfies the following equations for all for all $n \in N^g$ with $\mathsf{lab}^g(n) \notin \Delta$:

$$\mathsf{lab}^g(n) = \mathsf{lab}^h(\phi(n)) \qquad \text{(labelling)}$$

$$\phi(\mathsf{suc}^g_i(n)) = \mathsf{suc}^h_i(\phi(n)) \quad \text{for all } 0 \le i < \mathsf{ar}(\mathsf{lab}^g(n)) \qquad \text{(successor)}$$

For $\Delta = \emptyset$, we get the usual notion of homomorphisms on term graphs and from that the notion of isomorphisms. Nodes labelled with symbols in Δ can be thought of as holes in the term graphs (e.g. variables or $\bot$). Homomorphisms also give us a way to describe differences in sharing: given two term graphs g and h, we say that g has *less sharing than* h, written $g \le^{\mathsf{S}} h$, if there is a homomorphism $\phi\colon g \to h$.

Canonical Form, Unravelling, Bisimilarity. We do not want to distinguish between isomorphic term graphs. Therefore, we use a well-known trick [13] to obtain canonical representatives of isomorphism classes: a term graph g is called *canonical* if $n = \mathcal{P}_g(n)$ holds for each $n \in N^g$. The set of all (finite) canonical term graphs over Σ is denoted $\mathcal{G}^\infty_{\mathcal{C}}(\Sigma)$ (resp. $\mathcal{G}_{\mathcal{C}}(\Sigma)$). For each term graph $h \in \mathcal{G}^\infty_{\mathcal{C}}(\Sigma)$, its *canonical representative* $\mathcal{C}(h)$ is obtained from h by replacing each node n in h by $\mathcal{P}_h(n)$.

We consider the set of terms $\mathcal{T}^\infty(\Sigma)$ as the subset of canonical term trees of $\mathcal{G}^\infty_{\mathcal{C}}(\Sigma)$. With this correspondence in mind, we can define the *unravelling* of a term graph g as the unique term $\mathcal{U}(g)$ such that there is a homomorphism $\phi\colon \mathcal{U}(g) \to g$. Two term graphs g, h are called *bisimilar*, denoted $g \simeq h$, if $\mathcal{U}(g) = \mathcal{U}(h)$.

Labelled Quotient Tree. We shall use an alternative representation that describes term graphs uniquely up to isomorphism. To this end, we define the binary relation $\sim_g$ on positions in the term graph g as follows: $\pi_1 \sim_g \pi_2$ iff $\mathsf{node}_g(\pi_1) = \mathsf{node}_g(\pi_2)$. That is, positions are related if they lead to the same node. The triple $(\mathcal{P}(g), g(\cdot), \sim_g)$, called *labelled quotient tree*, consisting of the mapping $g(\cdot)\colon \mathcal{P}(g) \to \Sigma$ and the binary relation $\sim_g$ over $\mathcal{P}(g)$ as defined

above characterises the term graph g up to isomorphism. In particular, each canonical term graph is uniquely determined by exactly one labelled quotient tree. The name is derived from the fact that $(\mathcal{P}(g), g(\cdot))$ describes a labelled tree and $\sim_g$ is a congruence on the set of nodes in this tree.

Metric Space. Next we present the partial order and the metric on term graphs that give us the modes of convergence needed for infinitary rewriting. The metric $\mathbf{d}_\dagger$ on term graphs is defined analogously to the metric that is used in infinitary term rewriting [8]. We define $\mathbf{d}_\dagger(g,h) = 0$ if $g = h$ and otherwise $\mathbf{d}_\dagger(g,h) = 2^{-d}$, where d is the minimal depth at which g and h differ. More precisely, d is defined as the largest number e such that g and h become isomorphic if all nodes at depth e are relabelled with a fresh symbol $\bot$ and their outgoing edges are removed (along with all nodes that become unreachable). We can give a concise characterisation of limits in the resulting metric space using labelled quotient trees:

▶ **Theorem 1** ([3, 4]). *$(\mathcal{G}^\infty_{\mathcal{C}}(\Sigma), \mathbf{d}_\dagger)$ is a complete ultrametric space, and the limit of each Cauchy sequence $(g_\iota)_{\iota<\alpha}$ is given by the labelled quotient tree $(P, l, \sim)$ with*

$$P = \liminf_{\iota\to\alpha} \mathcal{P}(g_\iota) = \bigcup_{\beta<\alpha} \bigcap_{\beta\le\iota<\alpha} \mathcal{P}(g_\iota) \qquad \sim = \liminf_{\iota\to\alpha} \sim_{g_\iota} = \bigcup_{\beta<\alpha} \bigcap_{\beta\le\iota<\alpha} \sim_{g_\iota}$$

$$l(\pi) = g_\beta(\pi) \quad \text{if } \exists\beta<\alpha\forall\beta\le\iota<\alpha\colon g_\iota(\pi) = g_\beta(\pi) \qquad \text{for all } \pi\in P$$

Intuitively, the limit of a Cauchy sequence $(g_\iota)_{\iota<\alpha}$ is the tree consisting of all nodes that become eventually stable in $(\mathcal{U}(g_\iota))_{\iota<\alpha}$ (i.e. remain unchanged in the unravelling from some point onwards), but quotiented to a graph by sharing all nodes that eventually remain shared in $(\mathcal{U}(g_\iota))_{\iota<\alpha}$. An example is depicted in Figure 1c.

Partial Order. To define a partial order on term graphs, we consider signatures of the form $\Sigma_\bot$ that extend a signature Σ with a fresh constant symbol $\bot$. We call term graphs over $\Sigma_\bot$ *partial*, and term graphs over Σ *total*. We then use Δ-homomorphisms with $\Delta = \{\bot\}$ – also called *$\bot$-homomorphisms* – to define the *simple partial order* $\le^{\mathsf{S}}_\bot$ on $\mathcal{G}^\infty_{\mathcal{C}}(\Sigma_\bot)$ as follows: $g \le^{\mathsf{S}}_\bot h$ iff there is a $\bot$-homomorphism $\phi\colon s \to_\bot t$. Using labelled quotient trees, we get the following alternative characterisation:

▶ **Corollary 2** (characterisation of $\le^{\mathsf{S}}_\bot$, [3, 4]). *Let $g, h \in \mathcal{G}^\infty_{\mathcal{C}}(\Sigma_\bot)$. Then $g \le^{\mathsf{S}}_\bot h$ iff, for all $\pi, \pi' \in \mathcal{P}(g)$, we have*

(a) $\pi \sim_g \pi' \implies \pi \sim_h \pi'$, *and*
(b) $g(\pi) = h(\pi)$ *if* $g(\pi) \in \Sigma$.

The partially ordered set $(\mathcal{G}^\infty_{\mathcal{C}}(\Sigma_\bot), \le^{\mathsf{S}}_\bot)$ forms a *complete semilattice*, i.e. it has a least element $\bot$, each directed set D in $(\mathcal{G}^\infty_{\mathcal{C}}(\Sigma_\bot), \le^{\mathsf{S}}_\bot)$ has a *least upper bound* (*lub*) $\bigsqcup D$, and every *non-empty* set B in $(\mathcal{G}^\infty_{\mathcal{C}}(\Sigma_\bot), \le^{\mathsf{S}}_\bot)$ has *greatest lower bound* (*glb*) $\bigsqcap B$. In particular, this means that for any sequence $(g_\iota)_{\iota<\alpha}$ in $(\mathcal{G}^\infty_{\mathcal{C}}(\Sigma_\bot), \le^{\mathsf{S}}_\bot)$, its *limit inferior*, defined by $\liminf_{\iota\to\alpha} g_\iota = \bigsqcup_{\beta<\alpha} \left(\bigsqcap_{\beta\le\iota<\alpha} g_\iota\right)$, exists.

▶ **Theorem 3** ([3, 4]). *$(\mathcal{G}^\infty_{\mathcal{C}}(\Sigma_\bot), \le^{\mathsf{S}}_\bot)$ is a complete semilattice. In particular, the limit inferior of a sequence $(g_\iota)_{\iota<\alpha}$ is given by the labelled quotient tree $(P, \sim, l)$ with*

$$P = \bigcup\nolimits_{\beta<\alpha} \{\pi \in \mathcal{P}(g_\beta) \mid \forall\pi' < \pi\forall\beta\le\iota<\alpha\colon g_\iota(\pi') = g_\beta(\pi')\}$$

$$\sim\ = (P\times P) \cap \bigcup\nolimits_{\beta<\alpha} \bigcap\nolimits_{\beta\le\iota<\alpha} \sim_{g_\iota}$$

$$l(\pi) = \begin{cases} g_\beta(\pi) & \text{if } \exists\beta<\alpha\forall\beta\le\iota<\alpha\colon g_\iota(\pi) = g_\beta(\pi) \\ \bot & \text{otherwise} \end{cases} \qquad \text{for all } \pi\in P$$

The limit inferior generalises the limit of Cauchy sequences to arbitrary sequences. Similarly to the limit, the limit inferior of a sequence $(g_\iota)_{\iota<\alpha}$ is also the tree consisting of all nodes that become eventually stable in $(\mathcal{U}(g_\iota))_{\iota<\alpha}$, quotiented to a graph by sharing nodes that become eventually shared in $(g_\iota)_{\iota<\alpha}$. But since $(g_\iota)_{\iota<\alpha}$ is not necessarily Cauchy, some nodes never become stable and are thus replaced by $\bot$-nodes in the limit inferior construction. For an example, consider the sequence $(g_\iota)_{\iota<\alpha}$ from Figure 1c, but where the label f is replaced by h in g_0, g_2, g_4 etc. The resulting sequence is not Cauchy anymore; its limit inferior can be obtained from g_ω in Figure 1c, by replacing the label f with $\bot$.

Another example of a complete semilattice is the prefix order $\leq$ on sequences, which allows us to generalise concatenation as follows: Let $(S_\iota)_{\iota<\alpha}$ be a sequence of sequences over some set A. The concatenation of $(S_\iota)_{\iota<\alpha}$, written $\prod_{\iota<\alpha} S_\iota$, is recursively defined as the empty sequence $\langle\rangle$ if $\alpha = 0$, $\left(\prod_{\iota<\beta} S_\iota\right) \cdot S_\beta$ if $\alpha = \beta + 1$, and $\bigsqcup_{\gamma<\alpha} \prod_{\iota<\gamma} S_\iota$ if α is a limit ordinal.

3 Term Graph Rewriting

In this paper, we adopt the term graph rewriting framework of Barendregt et al. [7]. To represent placeholders in rewrite rules, we use variables – in a manner that is very similar to term rewrite rules. To this end, we consider a signature $\Sigma_\mathcal{V} = \Sigma \uplus \mathcal{V}$ that extends the signature Σ with a countably infinite set $\mathcal{V}$ of nullary variable symbols.

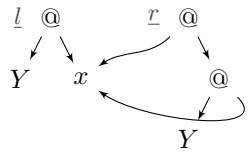
▶ **Definition 4** (term graph rewriting systems)**.** Given a signature Σ, a *term graph rule* ρ over Σ is a triple (g, l, r) where g is a graph over $\Sigma_\mathcal{V}$ and $l, r \in N^g$ such that all nodes in g are reachable from l or r. We write ρ_l and ρ_r to denote the left- and right-hand side of ρ, respectively, i.e. the term graph $g|_l$ and $g|_r$, respectively. Additionally, we require that for each variable $v \in \mathcal{V}$ there is at most one node n in g labelled v and that n is different from l but still reachable from l. ρ is called *left-linear* (resp. *left-finite*) if ρ_l is a term tree (resp. is finite). A *term graph rewriting system (GRS)* $\mathcal{R}$ is a pair (Σ, R) consisting of a signature Σ and a set R of term graph rules over Σ. $\mathcal{R}$ is called left-linear (resp. left-finite) if each rule of $\mathcal{R}$ is left-linear (resp. left-finite).

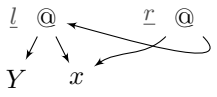
The requirement that the root l of the left-hand side is not labelled with a variable symbol is analogous to the requirement that the left-hand side of a term rule is not a variable. Similarly, the restriction that nodes labelled with variable symbols must be reachable from the root of the left-hand side corresponds to the restriction on term rewrite rules that every variable occurring on the right-hand side must also occur on the left-hand side.

▶ **Example 5.** Figure 1a shows two term graph rules which both unravel to the term rule $\rho\colon Y\,x \to x\,(Y\,x)$. Note that sharing of nodes is used both to refer from the right-hand side to variables on the left-hand side, and in order to simulate duplication.

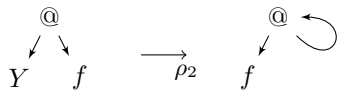
The notion of unravelling term graphs to terms straightforwardly extends to term graph rules: The *unravelling* of a term graph rule ρ, denoted $\mathcal{U}(\rho)$, is the term rule $\mathcal{U}(\rho_l) \to \mathcal{U}(\rho_r)$. The unravelling of a GRS $\mathcal{R} = (\Sigma, R)$, denoted $\mathcal{U}(\mathcal{R})$, is the TRS $(\Sigma, \{\mathcal{U}(\rho) \mid \rho \in R\})$.

Without going into all details of the construction, we describe the application of a rewrite rule ρ with root nodes l and r to a term graph g in four steps: at first a suitable sub-term graph $g|_n$ of g rooted in some node n of g is *matched* against the left-hand side of ρ. This matching amounts to finding a $\mathcal{V}$-homomorphism ϕ from the left-hand side ρ_l to $g|_n$. The term graph $g|_n$ is called a *redex*, and the pair (n, ρ) is called a *redex occurrence* in g; abusing notation we write (π, ρ) for the redex occurrence $(\mathsf{node}_g(\pi), \rho)$. The $\mathcal{V}$-homomorphism ϕ

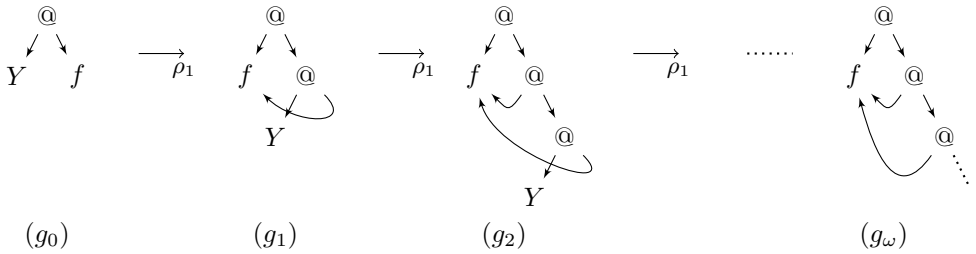




(a) Term graph rules that unravel to $Y\,x \to x\,(Y\,x)$.



(b) A single ρ_2-step.



(c) A strongly m-convergent term graph reduction over ρ_1.

Figure 1 Implementation of the fixed point combinator as a term graph rewrite rule.

instantiates variables in the rule with sub-term graphs of the redex. In the second step, nodes and edges in ρ that are not in ρ_l are copied into g, such that each edge pointing to a node m in ρ_l is redirected to $\phi(m)$. In the next step, all edges pointing to the root n of the redex are redirected to the root n' of the *contractum*, which is either r or $\phi(r)$, depending on whether r has been copied into g or not (because it is reachable from l in ρ). Finally, all nodes not reachable from the root of (the now modified version of) g are removed. With h the result of the this construction, we obtain a *pre-reduction step* $\psi\colon g \mapsto_{n,\rho,n'} h$ from g to h.

Figure 1b and 1c illustrate how the two rules in Figure 1a are applied to a term graph.

In order to define convergence on infinite reductions, we require that all term graphs are in canonical form. Therefore, we define a reduction step as a pre-reduction step as described above, where both term graphs have been turned into their canonical form:

▶ **Definition 6** (reduction steps). Let $\mathcal{R} = (\Sigma, R)$ be GRS, $\rho \in R$ and $g, h \in \mathcal{G}^\infty_{\mathcal{C}}(\Sigma)$ with $n \in N^g$ and $m \in N^h$. A tuple $\phi = (g, n, \rho, m, h)$ is called a *reduction step*, written $\phi\colon g \to_{n,\rho,m} h$, if there is a pre-reduction step $\phi'\colon g' \mapsto_{n',\rho,m'} h'$ with $\mathcal{C}(g') = g$, $\mathcal{C}(h') = h$, $n = \mathcal{P}_{g'}(n')$, and $m = \mathcal{P}_{h'}(m')$. We use the shorthand notation $\phi\colon g \to_{n,\rho} h$ and $\phi\colon g \to_n h$ if $\phi\colon g \to_{n,\rho,m} h$ for some m (and ρ). We write $\phi\colon g \to_{\mathcal{R}} h$ to indicate $\mathcal{R}$.

In this paper, we focus on the strong variant of convergence [3]. This variant of convergence takes into account the position of contracted redexes. For metric convergence, only the depth of the contracted redex is needed; for the partial order variant, we need an appropriate notion of reduction contexts, which is provided with the help of local truncations:

▶ **Definition 7** (local truncation). Let $g \in \mathcal{G}^\infty(\Sigma_\perp)$ and $M \subseteq N^g$. The *local truncation* of g at M, denoted $g\backslash M$, is obtained from g by labelling all nodes in M with $\perp$ and removing all outgoing edges from nodes in M (also removing all nodes that thus become unreachable from the root). Instead of $g\backslash\{n\}$ and $g\backslash\{\mathsf{node}_g(\pi)\}$, we also write $g\backslash n$ and $g\backslash\pi$, respectively.

Most of the time we will use the characterisation of local truncations in terms of labelled quotient trees instead of the definition above:

▶ **Lemma 8** ([3]). *For each $g \in \mathcal{G}^\infty(\Sigma_\bot)$ and $M \subseteq N^g$, the local truncation $g\backslash M$ has the following labelled quotient tree $(P, l, \sim)$:*

$$P = \{\pi \in \mathcal{P}(g) \mid \forall \pi' < \pi \colon \mathsf{node}_g(\pi') \notin M\} \qquad l(\pi) = \begin{cases} g(\pi) & \text{if } \mathsf{node}_g(\pi) \notin M \\ \bot & \text{if } \mathsf{node}_g(\pi) \in M \end{cases}$$
$$\sim = \sim_g \cap\, P \times P$$

Now we have everything in place to define our notions of convergence:

▶ **Definition 9** ([3]). Let $\mathcal{R} = (\Sigma, R)$ be a GRS.

(i) The *reduction context* c of a graph reduction step $\phi \colon g \to_n h$ is the term graph $\mathcal{C}(g\backslash n)$. We write $\phi \colon g \to_c h$ to indicate the reduction context c.

(ii) A *reduction* in $\mathcal{R}$, is a sequence $(\phi_\iota \colon g_\iota \to_\mathcal{R} g_{\iota+1})_{\iota<\alpha}$ of rewrite steps in $\mathcal{R}$.

(iii) Let $S = (\phi_\iota \colon g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\alpha}$ be a reduction in $\mathcal{R}$. S is *m-continuous* in $\mathcal{R}$ if $\lim_{\iota\to\lambda} g_\iota = g_\lambda$ and $(\mathsf{depth}_{g_\iota}(n_\iota))_{\iota<\lambda}$ tends to infinity for each limit ordinal $\lambda < \alpha$. S *m-converges* to g in $\mathcal{R}$, denoted $S \colon g_0 \xrightarrow{m}_\mathcal{R} g$, if it is m-continuous and either S is closed with $g = g_\alpha$ or S is open with $g = \lim_{\iota\to\alpha} g_\iota$ and $(\mathsf{depth}_{g_\iota}(n_\iota))_{\iota<\alpha}$ tending to infinity.

(iv) Let $S = (\phi_\iota \colon g_\iota \to_{c_\iota} g_{\iota+1})_{\iota<\alpha}$ be a reduction in $\mathcal{R}_\bot = (\Sigma_\bot, R)$. S is *p-continuous* in $\mathcal{R}$ if $\liminf_{\iota\to\lambda} c_\iota = g_\lambda$ for each limit ordinal $\lambda < \alpha$. S *p-converges* to g in $\mathcal{R}$, denoted $S \colon g_0 \xrightarrow{p}_\mathcal{R} g$, if it is p-continuous and either S is closed with $g = g_\alpha$ or S is open with $g = \liminf_{\iota\to\alpha} c_\iota$.

Note that we have to extend the signature of $\mathcal{R}$ to $\Sigma_\bot$ for the definition of p-convergence. We obtain the *total fragment* of p-convergence if we restrict ourselves to total term graphs: A reduction $(g_\iota \to_{\mathcal{R}_\bot} g_{\iota+1})_{\iota<\alpha}$ p-converging to g is called p-converging to g in $\mathcal{G}^\infty_\mathcal{C}(\Sigma)$ if g as well as each g_ι is total, i.e. $\{g_\iota \mid \iota < \alpha\} \cup \{g\} \subseteq \mathcal{G}^\infty_\mathcal{C}(\Sigma)$.

We have the following correspondence between m- and p-convergence:

▶ **Theorem 10** ([3]). *Let $\mathcal{R}$ be a GRS and S a reduction in $\mathcal{R}_\bot$. We then have that*

$$S \colon g \xrightarrow{m}_\mathcal{R} h \qquad \textit{iff} \qquad S \colon g \xrightarrow{p}_\mathcal{R} h \text{ in } \mathcal{G}^\infty_\mathcal{C}(\Sigma)\,.$$

Most of our results will be restricted to GRSs with (weakly) non-overlapping rules:

▶ **Definition 11** ((weakly) non-overlapping, [7]). Let (n, ρ) and (n', ρ') be redex occurrences in a term graph g, with corresponding matching $\mathcal{V}$-homomorphisms $\phi \colon \rho_l \to g|_n$ and $\phi' \colon \rho'_l \to g|_{n'}$.

(i) (n, ρ) and (n', ρ') are called *disjoint* if $n' \notin \phi(N)$ and $n \notin \phi'(N')$, where N and N' are the non-variable nodes in ρ_l and ρ'_l, respectively.

(ii) (n, ρ) and (n', ρ') are called *weakly disjoint* if they are either (a) disjoint, or (b) $n = n'$ and contracting both redexes results in isomorphic term graphs, i.e. $g \mapsto_{n,\rho} h \cong h' \mathrel{\reflectbox{$\mapsto$}}_{n',\rho'} g$.

A GRS $\mathcal{R}$ is *non-overlapping* resp. *weakly non-overlapping* if for every term graph g in $\mathcal{R}$, every two distinct redex occurrences are disjoint resp. weakly disjoint. A GRS that is non-overlapping and left-linear is called *orthogonal*.

Below we summarise the soundness and completeness property of infinitary term graph rewriting in terms of infinitary term rewriting.

▶ **Theorem 12** (soundness & completeness, [3]). *Let $\mathcal{R}$ be a left-finite GRS.*

(i) *If $\mathcal{R}$ is left-linear, then $g \xrightarrow{p}_\mathcal{R} h$ implies $\mathcal{U}(g) \xrightarrow{p}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$.*

(ii) *If $\mathcal{R}$ is orthogonal, then $\mathcal{U}(g) \xrightarrow{p}_{\mathcal{U}(\mathcal{R})} t$ implies $g \xrightarrow{p}_\mathcal{R} h$ and $t \xrightarrow{p}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$.*

The complete semilattice structure that underlies the definition of p-convergence ensures that every p-continuous reduction also p-converges – a property that distinguishes it from m-convergence. In other words, any open well-formed reduction can be uniquely completed to a closed well-formed reduction in the partial order model. A consequence of Theorem 10 is that a p-convergent reduction that does not m-converges must produce nodes labelled $\bot$. In the following, we analyse this formation of $\bot$-nodes and characterise it in terms of volatile positions, which are positions repeatedly contracted in a reduction:

▶ **Definition 13** (volatility). Let $S = (g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\lambda}$ be an open p-converging reduction. A position π is said to be *volatile* in S if, for each $\alpha < \lambda$, there is some $\alpha \leq \beta < \lambda$ such that $\pi \in n_\beta$. If π is volatile and no proper prefix of it is volatile in S, then π is called *outermost-volatile* in S.

Moreover, we need to characterise positions that are affected by rewrite steps:

▶ **Definition 14.** Let π be a position and n a node in a term graph g. Then π is said to *pass through* n in g if there is a prefix $\pi' \leq \pi$ with $\pi' \in \mathcal{P}_g(n)$, and π is said to *properly pass through* n in g if there is a proper prefix $\pi' < \pi$ with $\pi' \in \mathcal{P}_g(n)$.

Using Lemma 8 and Theorem 3 we can give the following characterisation of the formation $\bot$-nodes, where $\mathcal{P}_{\not\bot}(g)$ denotes the positions of nodes in a term graph $g \in \mathcal{G}^\infty(\Sigma_\bot)$ that are not labelled with $\bot$:

▶ **Lemma 15** (volatility). *Let $S = (g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\lambda}$ be an open reduction p-converging to g_λ. Then, for every position π, we have the following:*

(i) *If π is volatile in S, then $\pi \notin \mathcal{P}_{\not\bot}(g_\lambda)$.*

(ii) *$g_\lambda(\pi) = \bot$ iff*

- **(a)** *π is outermost-volatile in S, or*
- **(b)** *there is some $\alpha < \lambda$ such that $g_\alpha(\pi) = \bot$ and, for all $\alpha \leq \iota < \lambda$, π does not pass through n_ι in g_ι.*

Volatile positions give us the vocabulary to formulate the following variant of Theorem 10:

▶ **Corollary 16.** *For every GRS $\mathcal{R}$, $g \in \mathcal{G}^\infty_\mathcal{C}(\Sigma)$, and reduction S in $\mathcal{R}_\bot$, we have that $S\colon g \overset{p}{\twoheadrightarrow}_\mathcal{R} h$ and no open prefix of S has a volatile position iff $S\colon g \overset{m}{\twoheadrightarrow}_\mathcal{R} h$.*

Proof. This follows straightforwardly from Theorem 10 using Lemma 15(ii). ◀

4 Residuals and Projections

In this section, we develop the theory of residuals and projections for infinitary term graph rewriting.[1] We then use this machinery to prove the infinitary strip lemma and the compression lemma for both p- and m-convergence. We start by recapitulating the basic definitions and properties of residuals and projections for single reduction steps from Barendregt et al. [7].

Given two disjoint redex occurrences (n, ρ) and (n', ρ') in a term graph g, with matching $\mathcal{V}$-homomorphisms ϕ and ϕ', respectively, and a pre-reduction step $g \mapsto_{n,\rho} h$, we know that either n' is not a node in h, or there is a redex occurrence (n', ρ') in h [7]. This finding motivates the definition of residuals and projections:

[1] This section is heavily abridged; see Appendix A for the full theory and all proofs.

▶ **Definition 17** (reduction step residuals,[7]). Let $\psi\colon g \to_{n,\rho} h$ be a reduction step, $\overline{\psi}\colon \overline{g} \mapsto_{\overline{n},\rho} \overline{h}$ the underlying pre-reduction step, and (n',ρ') a redex occurrence in g weakly disjoint from (n,ρ); let $\overline{n'}$ be the node corresponding to n' in $\overline{g}$, i.e. $\overline{n'} = \phi(n')$, where ϕ is the isomorphism from g to $\overline{g}$.

(i) The *residual* of (n',ρ') by ψ, denoted $(n',\rho')//\psi$, is either
- **(a)** the empty set $\emptyset$ if (n',ρ') and (n,ρ) are not disjoint or $\overline{n'} \notin N^{\overline{h}}$, or
- **(b)** $\mathcal{P}_{\overline{h}}(\overline{n}')$ if (n',ρ') and (n,ρ) are disjoint and $\overline{n'} \in N^{\overline{h}}$.

(ii) The *projection* of the reduction step $\psi'\colon g \to_{n',\rho'} h'$ by ψ, denoted ψ'/ψ, is either
- **(a)** the empty reduction if $(n',\rho')//\psi = \emptyset$, or
- **(b)** the single step reduction contracting the ρ-redex rooted in $(n',\rho')//\psi$ in h otherwise.

Note that the residual $(n',\rho')//\psi$ is either the empty set or a node in h, namely $\mathcal{P}_{\overline{h}}(\overline{n}')$. This property generalises to residuals by reductions of arbitrary length:

▶ **Definition 18** (residuals). Let $\mathcal{R}$ be a weakly non-overlapping GRS, $S\colon g_0 \overset{p}{\twoheadrightarrow}_{\mathcal{R}} g_\alpha$, and (n,ρ) a redex occurrence in g_0 with ρ a rule in $\mathcal{R}$. The *residual* of (n,ρ) by S, denoted $(n,\rho)//S$, is inductively defined on the length of S as follows:

- S is empty: $(n,\rho)//S = n$
- $S = T \cdot \langle\psi\rangle$: $(n,\rho)//S = \begin{cases} \emptyset & \text{if } (n,\rho)//T = \emptyset \\ (m,\rho)//\psi & \text{if } (n,\rho)//T = m \neq \emptyset \end{cases}$
- S is open: $(n,\rho)//S = \mathcal{P}_{\not\perp}(g_\alpha) \cap \liminf_{\iota\to\alpha}(n,\rho)//S|_\iota$,
  that is $\quad \pi \in (n,\rho)//S \quad$ iff $\quad \pi \in \mathcal{P}_{\not\perp}(g_\alpha)$ and $\exists\beta < \alpha \forall \beta \leq \iota < \alpha\colon \pi \in (n,\rho)//S|_\iota$.

Note that since m-convergence is just a special case of p-convergence, according to Theorem 10, the definition of residuals also applies to m-convergent reductions. For *open* m-convergent reductions, however, we can simplify the characterisation by omitting the explicit requirement that a residual position has to be in $\mathcal{P}_{\not\perp}(g_\alpha)$.

Likewise, we can also generalise the notion of projections (cf. Figure 2). The basis for this generalisation is that given a reduction $S\colon g_0 \overset{p}{\twoheadrightarrow}_{\mathcal{R}} g_\alpha$ in a weakly non-overlapping GRS $\mathcal{R}$, and (n,ρ) a redex occurrence in g_0, we have that if $(n,\rho)//S = m$ is non-empty, then (m,ρ) is a redex occurrence in g_α:

▶ **Definition 19** (projections). Let $\mathcal{R}$ be a weakly non-overlapping GRS, $\phi\colon g \to_{n,\rho} h$ a reduction step in $\mathcal{R}$, and $S = (\psi_\iota\colon g_\iota \to g_{\iota+1})_{\iota<\alpha}$ a p-converging reduction in $\mathcal{R}$. The *projection* of ϕ by S, denoted ϕ/S, is

(a) the empty reduction if $(n,\rho)//S = \emptyset$, and

(b) the single step reduction contracting the ρ-redex rooted in $(n,\rho)//S$ in h otherwise.

The *projection* of S by ϕ, denoted S/ϕ, is defined as the concatenation $\prod_{\iota<\alpha} \psi_\iota/\,(\phi/S|_\iota)$.

One can show that projections commute for both m- and p-convergent reductions given that one of the reductions is finite:

▶ **Theorem 20** (infinitary strip lemma: p-convergence). *Let $\mathcal{R}$ be a weakly non-overlapping GRS, $\phi\colon g_0 \to_{n,\rho} h_0$ a reduction step in $\mathcal{R}$, $S\colon g_0 \overset{p}{\twoheadrightarrow}_{\mathcal{R}} g_\alpha$, and $\phi/S\colon g_\alpha \to^{\leq 1}_{\mathcal{R}} h_\alpha$. Then we have that $S/\phi\colon h_0 \overset{p}{\twoheadrightarrow}_{\mathcal{R}} h_\alpha$.*

Note that the strip lemma for term graph rewriting is simpler than for term rewriting as a redex has at most one residual and, thus, we do not have to deal with complete developments. The proof of the strip lemma constructs the commuting diagram shown in Figure 2. For the basic squares we can use the result of Barendregt et al. [7] who showed that projections of single reduction steps commute.

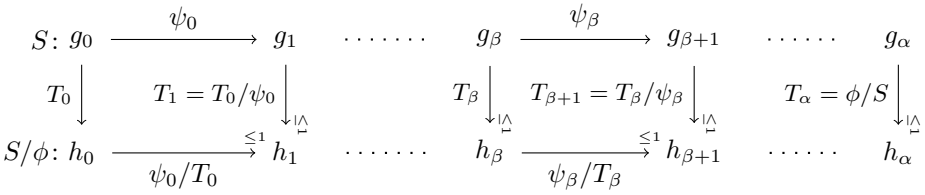


Figure 2 The Infinitary Strip Lemma.

From the above infinitary strip lemma, one can derive the corresponding variant for m-convergence using Corollary 16 fairly easily:

▶ **Theorem 21** (infinitary strip lemma: m-convergence)**.** *Let $\mathcal{R}$ be a weakly non-overlapping GRS, $\phi\colon g_0 \to_{n,\rho} h_0$ a reduction step in $\mathcal{R}$, $S\colon g_0 \xrightarrow{m}_{\mathcal{R}} g_\alpha$, and $\phi/S\colon g_\alpha \to^{\leq 1} h_\alpha$. Then we have that $S/\phi\colon h_0 \xrightarrow{m}_{\mathcal{R}} h_\alpha$.*

The definition of projections can be generalised to projections S/T of arbitrary pairs of reductions S, T in the obvious way (by extending Figure 2 vertically). While we conjecture that these general projections of p-convergent reductions commute as well, which means that we have infinitary confluence, the same cannot be said for m-convergence: the counterexample of Kennaway et al. [9] applies here as well.

The infinitary strip lemmas are a powerful tool as we shall see. Below we will apply them to prove that reductions can be compressed to length at most ω – a useful property in its own right. To this end, we need the two lemmas below. The first one states that any redex obtained by an open reduction must already occur in an earlier term graph, which is subsequently unaffected by reduction. The second lemma states that also all positions within the redex itself remain untouched.

▶ **Lemma 22.** *Given an open reduction $S = (g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\lambda}$ p-converging to g_λ and a redex occurrence (π, ρ) in g_λ with ρ left-finite, there is a position $\pi \in \mathcal{P}(g_\lambda)$ and some $\alpha < \lambda$ such that (π, ρ) is a redex occurrence in g_ι, and π does not pass through n_ι in g_ι for any $\alpha \leq \iota < \lambda$.*

▶ **Lemma 23.** *Let $S\colon g \xrightarrow{p} h$ be a p-converging reduction in a weakly non-overlapping GRS $\mathcal{R}$ and (n, ρ) a redex occurrence in g. For each $\pi \in \mathcal{P}_g(n)$ such that π does not pass through the root of any redex contracted in S, we have that $\pi \in (n, \rho)//S$.*

Proof. Straightforward induction on the length of S. ◀

The proof of the full compression property for p-convergent reductions is tricky. For now, we only show that infinite, closed reductions can be compressed. This property will turn out to be sufficient for our purposes and later in Section 5, we can use our main result to extend it to full compression much more easily.

▶ **Proposition 24** (compression of closed transfinite reductions)**.** *Let $S\colon g \xrightarrow{p}_{\mathcal{R}} h$ in a weakly non-overlapping, left-finite GRS $\mathcal{R}$. Then there is a reduction $T\colon g \xrightarrow{p}_{\mathcal{R}} h$ that is finite or open but not longer than S.*

Proof Sketch. We proceed by induction on the length of S. The only non-trivial case is where $S = S' \cdot \langle\phi\rangle$ with $S'\colon g \xrightarrow{p} g'$ and $\phi\colon g' \to h$. By induction hypothesis there is a finite or open reduction $T'\colon g \xrightarrow{p} g'$ of length at most $|S'|$. If T' is finite, then so is $T' \cdot \langle\phi\rangle\colon g \xrightarrow{p} h$. Otherwise, let (π, ρ) be the redex occurrence contracted in ϕ and construct

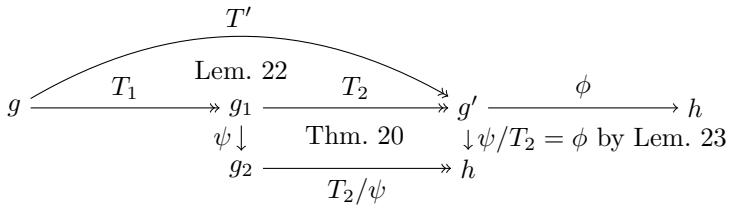


Figure 3 Compression of closed transfinite reductions.

the diagram illustrated in Figure 3, where ψ contracts (π, ρ) in g_1. This gives us a reduction $T_3 = T_1 \cdot \langle\psi\rangle \cdot T_2/\psi$ with $T_3\colon g \overset{p}{\twoheadrightarrow} h$ and $|T_3| < |S|$. Thus, we may apply the induction hypothesis to T_3 to obtain a finite or open reduction $T\colon g \overset{p}{\twoheadrightarrow} h$ ◀

The above proof carries over to m-convergent reductions by using Theorem 21 instead of Theorem 20. Moreover, we can strengthen it to obtain full compression for m-convergent reductions:

▶ **Proposition 25.** *Let $S\colon g \overset{m}{\twoheadrightarrow} h$ in a weakly non-overlapping, left-finite GRS. Then there is a reduction $T\colon g \overset{m}{\twoheadrightarrow} h$ of length at most ω.*

Proof. By the proof of Lemma 5.1 in [9] it suffices to show the property for $|S| = \omega+1$, which can be done analogously to Proposition 24 but using Theorem 21 instead of Theorem 20. ◀

We conclude this section by deriving a compression property for reductions p-converging to $\bot$. To this end, we need the following lemma, which states that any term graph that reduces to $\bot$ must also reduce to a redex:

▶ **Lemma 26.** *For each reduction $S\colon g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} \bot$ in a weakly non-overlapping, left-finite GRS $\mathcal{R}$ with $g \neq \bot$, we find a finite reduction $g \to^*_{\mathcal{R}} h$ to a redex h.*

Proof Sketch. There is at least one step in S contracting a redex at the root, i.e. a proper prefix T of S p-converges to a redex. By induction on the length of T, we show that there is a finite reduction from g to a redex: By Proposition 24, we may assume that T is finite or open. If T is finite, we are done. Otherwise, we use Lemma 22 to find a proper prefix of T that p-converges to a redex. The induction hypothesis then yields the finite reduction to a redex. ◀

Given this property we can compress any reduction to $\bot$ to a length of at most ω:

▶ **Proposition 27.** *For each reduction $S\colon g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} \bot$ in a weakly non-overlapping, left-finite GRS $\mathcal{R}$, there is a reduction $T\colon g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} \bot$ of length at most ω.*

Proof. Let $g_0 \overset{p}{\twoheadrightarrow}_{\mathcal{R}} \bot$ with $g_0 \neq \bot$. Then we may apply Lemma 26 to obtain a reduction $g_0 \to^* h_0 \to g_1$ whose last rewrite step is at the root. By Theorem 20, there is also a reduction $g_1 \overset{p}{\twoheadrightarrow} \bot$. Hence, we may repeat this construction to obtain a reduction of the form $g_0 \to^* h_0 \to g_1 \to^* h_1 \to g_2 \to^* \dots$ Either the construction stops at some $i < \omega$ because $g_i = \bot$ in which case we have found a finite reduction $T\colon g_0 \to^* \bot$, or there is no such i with $g_i = \bot$ and we have found a reduction T of length ω with a volatile position $\langle\rangle$. Hence, $T\colon g_0 \overset{p}{\twoheadrightarrow} \bot$ according to Lemma 15. ◀

5 Böhm Reduction

Recall Theorem 10, which states that p-convergence and m-convergence coincide if we restrict ourselves to total term graphs. In this section, we show that the remaining gap between p- and m-convergence is bridged by adding rewrite rules that contract certain term graphs directly to $\bot$, thereby simulating reductions of the form $g \xrightarrow{p}\!\!\!\!\twoheadrightarrow \bot$. We give two characterisations of such term graphs:

▶ **Definition 28.** Let $\mathcal{R}$ be a GRS. A partial term graph g in $\mathcal{R}$ is called *fragile* if there is an open reduction $S\colon g \xrightarrow{p}\!\!\!\!\twoheadrightarrow_{\mathcal{R}} \bot$. A total term graph g in $\mathcal{R}$ is called *root-active* if for each reduction $g \to^*_{\mathcal{R}} h$ there is a reduction $h \to^*_{\mathcal{R}} h'$ to a redex h'. We write $\mathcal{RA}^{\mathcal{R}}$, or simply $\mathcal{RA}$, to denote the set of root-active terms in $\mathcal{R}$.

As it turns out the above two concepts – fragility and root-activeness – coincide on total term graphs. The following observation will help us to establish that:

▶ **Corollary 29.** *A total term graph g in a weakly non-overlapping, left-finite GRS $\mathcal{R}$ is fragile in $\mathcal{R}$ iff there is a reduction $g \xrightarrow{p}\!\!\!\!\twoheadrightarrow_{\mathcal{R}} \bot$.*

Proof. The "only if" direction follows by definition, whereas the "if" direction follows from Proposition 27 and the fact that total term graphs cannot reduce to $\bot$ in finitely many steps. ◀

▶ **Proposition 30.** *Let g be a total term graph in a weakly non-overlapping, left-finite GRS $\mathcal{R}$. Then g is root-active iff g is fragile.*

Proof. If g is root-active, then we can construct a reduction of length ω that infinitely often contracts a redex at the root and thus p-converges to $\bot$. For the converse direction assume some finite reduction $g \to^* h$. If g is fragile, then there is a reduction $g \xrightarrow{p}\!\!\!\!\twoheadrightarrow_{\mathcal{R}} \bot$, according to Corollary 29. By iterating Theorem 20, we thus find a reduction $h \xrightarrow{p}\!\!\!\!\twoheadrightarrow_{\mathcal{R}} \bot$. Moreover, since g is total, so is h. Hence, by Corollary 29, h is fragile, too. That means, according to Lemma 26 that there is a finite reduction from h to a redex. ◀

To bridge the gap between p- and m-convergence, we adopt the notion of Böhm extensions from term rewriting [10], which is a construction that extends TRSs by rules of the form $t \to \bot$. The definition on GRS is analogous:

▶ **Definition 31** (Böhm extension)**.** Let $\mathcal{R} = (\Sigma, R)$ be a GRS, and $\mathcal{U} \subseteq \mathcal{G}^\infty_{\mathcal{C}}(\Sigma)$.

(i) A *$\mathcal{U}$-instance* of a term graph $h \in \mathcal{G}^\infty_{\mathcal{C}}(\Sigma_\bot)$ is a term graph $g \in \mathcal{G}^\infty_{\mathcal{C}}(\Sigma)$ that is obtained from h by replacing each occurrence of $\bot$ in g with some term graph in $\mathcal{U}$, i.e. there is a set $M \subseteq N^g$ with $g|_m \in \mathcal{U}$ for all $m \in M$, and $h \cong g\backslash M$.

(ii) $\mathcal{U}_\bot$ is the set of term graphs in $\mathcal{G}^\infty_{\mathcal{C}}(\Sigma_\bot)$ that have a $\mathcal{U}$-instance in $\mathcal{U}$. In other words, $\mathcal{U}_\bot$ contains all those term graphs that can be obtained by taking a term graph g from $\mathcal{U}$ and replacing some sub-term graphs of g that are themselves in $\mathcal{U}$ with $\bot$.

(iii) The *Böhm extension* of $\mathcal{R}$ w.r.t. $\mathcal{U}$ is the GRS $\mathcal{B} = (\Sigma_\bot, R \cup B)$, where

$$B = \left\{ (g \uplus \bot, r^g, r^\bot) \,\middle|\, g \in \mathcal{U}_\bot \setminus \{\bot\} \right\}.$$

That is, B consists of rules with left-hand side $g \in \mathcal{U}_\bot \setminus \{\bot\}$ and right-hand side $\bot$. The rules in B are called *$\bot$-rules w.r.t. $\mathcal{U}$* and we write $g \to_\bot h$ for a reduction step using such a rule in B and call it a *$\bot$-step*.

In the remainder of this section we prove that $g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} h$ is equivalent to $g \overset{m}{\twoheadrightarrow}_{\mathcal{B}} h$, where $\mathcal{B}$ is the Böhm extension of $\mathcal{R}$ w.r.t. $\mathcal{RA}^{\mathcal{R}}$.

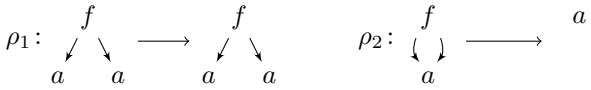
The semantics of term graph rewriting makes the behaviour of Böhm extensions slightly different compared to term rewriting. Not only term graphs in $\mathcal{U}_\perp$ are contracted to $\perp$ but also term graphs that have more sharing than those in $\mathcal{U}_\perp$:

▶ **Lemma 32.** *Let $g \to_{n,\rho,m} h$ be a reduction step of a $\perp$-rule ρ w.r.t. a set of term graphs $\mathcal{U} \subseteq \mathcal{G}^\infty_{\mathcal{C}}(\Sigma)$. Then there is some $g' \in \mathcal{U}_\perp \setminus \{\perp\}$ with $g' \leq^{\mathsf{S}} g|_n$ and $h = g\backslash n$.*

Proof. The equality $h = g\backslash n$ follows from fact that the right-hand side of ρ is by definition $\perp$. Since the rewrite step takes place at node n in g, we find a matching $\mathcal{V}$-homomorphism $\phi\colon \rho_l \to_{\mathcal{V}} g|_n$. By definition of $\perp$-rules, the left-hand side ρ_l of ρ is some term graph $g' \in \mathcal{U}_\perp \setminus \{\perp\}$. Hence, $\phi\colon g' \to_{\mathcal{V}} g|_n$. Since term graphs in $\mathcal{U}$ do not contain variables, g' does not contain variables either. Therefore, ϕ is a homomorphism. Consequently, $g' \leq^{\mathsf{S}} g|_n$. ◀

In general, this is a problem as root-active term graphs are not closed under increase of sharing. Consider the following example:

▶ **Example 33.**



In the GRS consisting of the two rules above, the left-hand side g of ρ_1 is root-active while the left-hand side h of ρ_2 is not, even though $g \leq^{\mathsf{S}} h$. However, if we consider orthogonal systems, this phenomenon cannot occur:

▶ **Lemma 34.** *Let $\mathcal{R}$ be an orthogonal, left-finite GRS and g, h two partial term graphs in $\mathcal{R}$ that are bisimilar. Then $g \overset{p}{\twoheadrightarrow} \perp$ iff $h \overset{p}{\twoheadrightarrow} \perp$.*

Proof. As bisimilarity is symmetric we only need to show one direction. Assume that $g \simeq h$ and that $g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} \perp$. By Theorem 12(i), we find a reduction $\mathcal{U}(g) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \perp$, since $\mathcal{U}(\perp) = \perp$. Since $g \simeq h$, we know that $\mathcal{U}(g) = \mathcal{U}(h)$, which means that $\mathcal{U}(h) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \perp$. Since $\perp$ is a normal form in $\mathcal{U}(\mathcal{R})$, we find, according to Theorem 12(ii), a reduction $h \overset{p}{\twoheadrightarrow}_{\mathcal{R}} \perp$. ◀

Thus, fragility and, by Proposition 30, root-activeness is preserved by bisimilarity. By a similar argument, we have preservation by p-converging reductions as well:

▶ **Lemma 35.** *Let $g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} h$ and $g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} \perp$ be a reduction in an orthogonal, left-finite GRS $\mathcal{R}$. Then there is a reduction $h \overset{p}{\twoheadrightarrow} \perp$.*

Proof. By Theorem 12(i), we have $\mathcal{U}(g) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$ and $\mathcal{U}(g) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \perp$. Since $\mathcal{R}$ is orthogonal and left-finite, so is $\mathcal{U}(\mathcal{R})$. Because orthogonal, left-finite TRSs are known to be infinitary confluent w.r.t. p-convergence [2], we know that there is a reduction $\mathcal{U}(h) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \perp$. Since $\perp$ is a normal form in $\mathcal{U}(\mathcal{R})$, we may apply Theorem 12(ii) to obtain a term graph reduction $h \overset{p}{\twoheadrightarrow}_{\mathcal{R}} \perp$. ◀

Next, we show that for each $\mathcal{RA}$-instance g of a term graph h, we have $g \overset{p}{\twoheadrightarrow} h$.

▶ **Lemma 36.** *If g is a total term graph in a GRS $\mathcal{R}$ that is an $\mathcal{RA}$-instance of a term graph h, then $g \overset{p}{\twoheadrightarrow} h$.*

Proof Sketch. We know that $h = g\backslash M$ for some set of nodes M and $g|_m \in \mathcal{RA}$ for all $m \in M$. We then construct a reduction $S\colon g_0 \overset{p}{\twoheadrightarrow}^{\omega} g_1 \overset{p}{\twoheadrightarrow}^{\omega} g_2 \overset{p}{\twoheadrightarrow}^{\omega} \dots g_\omega$ starting in $g_0 = g$ and p-converging to g_ω. Each reduction $g_i \overset{p}{\twoheadrightarrow}^{\omega} g_{i+1}$ in S rewrites a root-active sub-term graph $g|_m$

to $\bot$. If M is finite, $g_\omega = h$ follows easily. Otherwise, one can show that $\liminf_{i\to\omega} g_{i+1} = g_\omega$, which implies $h \leq^{\mathsf{S}}_{\bot} g_\omega$ since $h \leq^{\mathsf{S}}_{\bot} g_i$ for all $i < \omega$. Using Corollary 2 we can then show with the help of Theorem 3 that $g_\omega \leq^{\mathsf{S}}_{\bot} h$. Hence $S\colon g \xrightarrow{p}\!\!\!\!\rightarrow h$. ◀

According to Proposition 30, each term graph $g \in \mathcal{RA}$ is characterised by a reduction $g \xrightarrow{p}\!\!\!\!\rightarrow \bot$. With the above lemma, this property generalises to $\mathcal{RA}_\bot$.

▶ **Proposition 37.** *In orthogonal, left-finite GRSs, we have $g \in \mathcal{RA}_\bot$ iff $g \xrightarrow{p}\!\!\!\!\rightarrow \bot$.*

Proof. If $g \in \mathcal{RA}_\bot$, then there is some $h \in \mathcal{RA}$ that is an $\mathcal{RA}$-instance of g. According to Lemma 36, we thus find a reduction $h \xrightarrow{p}\!\!\!\!\rightarrow g$. By Proposition 30, there is a reduction $h \xrightarrow{p}\!\!\!\!\rightarrow \bot$. Applying Lemma 35, we find a reduction $g \xrightarrow{p}\!\!\!\!\rightarrow \bot$.

For the converse direction we show that if $g \xrightarrow{p}\!\!\!\!\rightarrow \bot$ and $h \in \mathcal{G}^\infty_{\mathcal{C}}(\Sigma)$ is an $\mathcal{RA}$-instance of g, then h is root-active. By Lemma 36, we find a reduction $g \xrightarrow{p}\!\!\!\!\rightarrow h$, which means, according to Lemma 35, that there is a reduction $h \xrightarrow{p}\!\!\!\!\rightarrow \bot$. By Corollary 29, we know that h is fragile, which implies, by Proposition 30, that h is root-active. ◀

Finally, we have everything in place to prove our main result:

▶ **Theorem 38.** *Let $\mathcal{R}$ be an orthogonal, left-finite GRS and $\mathcal{B}$ its Böhm extension w.r.t. $\mathcal{RA}$. Then we have that $g \xrightarrow{p}\!\!\!\!\rightarrow_{\mathcal{R}} h$ iff $g \xrightarrow{m}\!\!\!\!\rightarrow_{\mathcal{B}} h$.*

Proof Sketch. $\mathcal{B}$ is a GRS over the signature $\Sigma' = \Sigma \uplus \{\bot\}$, i.e. term graphs containing $\bot$ are considered total in $\mathcal{B}$, which justifies our use of Corollary 16 and Theorem 10 below.

Given a reduction $S\colon g \xrightarrow{m}\!\!\!\!\rightarrow_{\mathcal{B}} h$, we know that, by Theorem 10, $S\colon g \xrightarrow{p}\!\!\!\!\rightarrow_{\mathcal{B}} h$, too. We construct a reduction T from S by replacing each $\bot$-step $\widehat{g} \to_{\bot,n} \widehat{h}$ in S by a reduction $S'\colon \widehat{g} \xrightarrow{p}\!\!\!\!\rightarrow_{\mathcal{R}} \widehat{h}$. For each such $\bot$-step there is, by Lemma 32, some $\overline{g} \in \mathcal{RA}_\bot \setminus \{\bot\}$ with $\overline{g} \leq^{\mathsf{S}} \widehat{g}|_n$ and $\widehat{h} = \widehat{g}\backslash n$. Hence, by Proposition 37 and Lemma 34, we find a reduction $\widehat{g}|_n \xrightarrow{p}\!\!\!\!\rightarrow_{\mathcal{R}} \bot$. By embedding this reduction in $\widehat{g}$ at node n, we obtain the desired reduction $S'\colon \widehat{g} \xrightarrow{p}\!\!\!\!\rightarrow_{\mathcal{R}} \widehat{h}$. Using Theorem 3, one can show that the thus obtained reduction T p-converges to h.

Given $S\colon g \xrightarrow{p}\!\!\!\!\rightarrow_{\mathcal{R}} h$, we construct a reduction $T\colon g \xrightarrow{p}\!\!\!\!\rightarrow_{\mathcal{B}} h$, without any volatile positions. For each open prefix $S|_\lambda$ with an outermost-volatile position π, we find some $\beta < \lambda$ such that no step between β and λ takes place strictly above π. We then remove all reduction steps between β and λ at π or below and replace them with a single $\bot$-step $g_\beta \to_\bot g'_\beta$, which is justified by Proposition 37 and Lemma 32. Using Lemma 15)(ii), one can show that the resulting reduction T p-converges to the same term graph h. By construction, no prefix of T contains a volatile position. Thus, we may apply Corollary 16 to conclude $T\colon g \xrightarrow{m}\!\!\!\!\rightarrow_{\mathcal{B}} h$. ◀

Using the above correspondence, we can leverage the compression property for m-converging reductions to obtain full compression for p-converging reductions:

▶ **Proposition 39.** *For every reduction $S\colon g \xrightarrow{p}\!\!\!\!\rightarrow_{\mathcal{R}} h$ in an orthogonal, left-finite GRS $\mathcal{R}$, there is a reduction $T\colon g \xrightarrow{p}\!\!\!\!\rightarrow_{\mathcal{R}} h$ of length at most ω.*

Proof Ssketch. According to Theorem 38, $g \xrightarrow{p}\!\!\!\!\rightarrow_{\mathcal{R}} h$ implies $g \xrightarrow{m}\!\!\!\!\rightarrow_{\mathcal{B}} h$. One can show that the latter reduction can be reordered to the form $g \xrightarrow{m}\!\!\!\!\rightarrow_{\mathcal{R}} g' \xrightarrow{m}\!\!\!\!\rightarrow_{\bot} h$ that performs the $\bot$-steps at the very end (cf. Lemma 27 from Kennaway et al. [10]). By Proposition 25 there is a reduction $S\colon g \xrightarrow{m}\!\!\!\!\rightarrow_{\mathcal{R}} g'$ of length at most ω. Moreover, we can show that there is a reduction $T\colon g' \xrightarrow{m}\!\!\!\!\rightarrow_{\bot} h$ of length at most ω (cf. Lemma 7.2.4 from Ketema [12]). As in the proof of Theorem 38, we can replace each application of a $\bot$-rule $r \to \bot$ in T with a reduction derived from a corresponding reduction $r \xrightarrow{p}\!\!\!\!\rightarrow_{\mathcal{R}} \bot$, which according to Proposition 27 has length at most ω. The thus obtained reduction $T'\colon g' \xrightarrow{p}\!\!\!\!\rightarrow_{\mathcal{R}} h$ has length at most $\omega \cdot \omega$. If S is finite,

then we interleave the reduction steps in T' to obtain a reduction $T'' \colon g' \overset{p}{\twoheadrightarrow}_{\mathcal{R}} h$ of length at most ω, and thus we get a reduction $S \cdot T'' \colon g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} h$ of length at most ω. Otherwise, if S is of length ω, then we can interleave the steps in T' into S as shown in the successor case of the proof of the Compression Lemma in [9] to obtain a reduction $g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} h$ of length ω. ◀

Using the above compression result, we can strengthen the correspondence result of Theorem 10 for orthogonal GRSs as follows:

▶ **Corollary 40.** *Given an orthogonal, left-finite GRS $\mathcal{R}$ and two total term graphs g, h in $\mathcal{R}$, we have $g \overset{m}{\twoheadrightarrow} h$ iff $g \overset{p}{\twoheadrightarrow} h$.*

Proof. The "only if" direction follows from Theorem 10. By Proposition 39, we may assume that $g \overset{p}{\twoheadrightarrow} h$ is not longer than ω. Since g is total, and totality is preserved by reduction steps, we may apply Theorem 10 to conclude that $g \overset{m}{\twoheadrightarrow} h$. ◀

That is, reachability between total term graphs is independent from the choice between m- and p-convergence.

6 Concluding Remarks

Böhm extensions already entail some technical complications in term rewriting, which require some care, e.g. the additional rewrite rules may have infinite left-hand sides (which breaks the precondition of the compression lemma for example). In term graph rewriting we get additional complications: a redex may have sharing that is different from the rule's left-hand side that it instantiates. This phenomenon motivated the restriction to left-linear systems as we illustrated in Example 33. However, we conjecture that the issue illustrated in Example 33 does not occur in weakly non-overlapping systems – making the left-linearity restriction superfluous.

For the proof of our main result in Section 5, we also moved from weakly non-overlapping to non-overlapping systems, which made it possible to leverage the soundness and completeness properties from Theorem 12 in the proofs of Lemma 34 and Lemma 35. We conjecture that this additional restriction is not essential and merely simplified the proof at these two points.

A question that remains unanswered is whether orthogonal GRSs are confluent w.r.t. p-convergence. We conjecture that this is the case, but the technical difficulties that we already encountered in the proof of the infinitary strip lemmas appear to multiply when analysing the general case of constructing a tiling diagram.

Note that confluence of p-converging term graph reductions *modulo bisimilarity* can be easily obtained using the soundness and completeness properties from Theorem 12. Given two reductions $g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} h_i$, $i \in \{1, 2\}$ in a left-finite orthogonal GRS $\mathcal{R}$, we have $\mathcal{U}(g) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h_i)$. Since $\mathcal{U}(\mathcal{R})$ is normalising and confluent w.r.t. p-convergence [2], we thus find reductions $\mathcal{U}(h_i) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} t$ to a normal form t. By completeness, we then have reductions $h_i \overset{p}{\twoheadrightarrow}_{\mathcal{R}} g_i$ with $\mathcal{U}(g_i) = t$, i.e. $g_1 \simeq g_2$. Due to the correspondence result of Theorem 38, this confluence property also carries over to m-convergence in the corresponding Böhm extension.

References

1 P. Bahr. Abstract models of transfinite reductions. In C. Lynch, editor, *RTA 10*, volume 6 of *LIPIcs*, pages 49–66, 2010. doi:10.4230/LIPIcs.RTA.2010.49.

2 P. Bahr. Partial order infinitary term rewriting and Böhm trees. In C. Lynch, editor, *RTA*, volume 6 of *LIPIcs*, pages 67–84, 2010. doi:10.4230/LIPIcs.RTA.2010.67.

3 P. Bahr. Infinitary term graph rewriting is simple, sound and complete. In A. Tiwari, editor, *RTA*, volume 15 of *LIPIcs*, pages 69–84, 2012. doi:10.4230/LIPIcs.RTA.2012.69.

4 P. Bahr. Convergence in infinitary term graph rewriting systems is simple. Unpublished manuscript, available from the author's website, 2013.

5 P. Bahr. Böhm reduction in infinitary term graph rewriting systems. Companion report, available from the author's website, 2017.

6 Henk P. Barendregt. Representing'undefined' in the lambda calculus. *Journal of Functional Programming*, 2:367–374, 1992.

7 H. P. Barendregt, M. C. J. D. van Eekelen, J. R. W. Glauert, R. Kennaway, M. J. Plasmeijer, and M. R. Sleep. Term graph rewriting. In *PARLE*, pages 141–158, 1987.

8 R. Kennaway and F.-J. de Vries. Infinitary rewriting. In Terese, editor, *Term Rewriting Systems*, chapter 12, pages 668–711. Cambridge University Press, 2003.

9 R. Kennaway, J. W. Klop, M. R. Sleep, and F.-J. de Vries. Transfinite reductions in orthogonal term rewriting systems. *Inf. Comput.*, 119(1):18–38, 1995. doi:DOI:10.1006/inco.1995.1075.

10 R. Kennaway, V. van Oostrom, and F.-J. de Vries. Meaningless terms in rewriting. *J. Funct. Logic Programming*, 1999(1):1–35, February 1999.

11 Richard Kennaway, Jan Willem Klop, M. R. Sleep, and Fer-Jan de Vries. Infinitary lambda calculus. *Theoretical Computer Science*, 175(1):93–125, 1997. doi:10.1016/S0304-3975(96)00171-5.

12 J. Ketema. *Böhm-Like Trees for Rewriting*. PhD thesis, Vrije Universiteit Amsterdam, 2006. URL: http://dare.ubvu.vu.nl/handle/1871/9203.

13 D. Plump. Term graph rewriting. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, pages 3–61. World Scientific Publishing Co., Inc., 1999.

A Residuals and Projections

In this appendix we give the full proofs for the theory of residuals and projections.

A.1 Preliminaries

In this section we list some properties from previous work [4] that are used for developing the theory of residuals and projections.

▶ **Lemma 41** ([4]). *Given $g, h \in \mathcal{G}^\infty(\Sigma)$, there is a $\phi\colon g \to_\Delta h$ iff for all $\pi, \pi' \in \mathcal{P}(g)$,*

$$\text{(a) } \pi \sim_g \pi' \implies \pi \sim_h \pi', \text{ and } \quad \text{(b) } g(\pi) = h(\pi) \quad \text{whenever} \quad g(\pi) \notin \Delta.$$

▶ **Definition 42** ([4]). A position $\pi \in \mathcal{P}(g)$ in a term graph $g \in \mathcal{G}^\infty(\Sigma)$ is called *redundant* if there are $\pi_1, \pi_2 \in \mathcal{P}(g)$ with $\pi_1 < \pi_2 < \pi$ such that $\pi_1 \sim_g \pi_2$. A position that is not redundant is called *essential*. The set of all essential positions of g are denoted $\mathcal{P}^e(g)$.

Intuitively, the set of essential positions of a term graph is a minimal set of positions that still describes its structure (up to isomorphism) completely. In particular, any repetition due to cycles is omitted. The following proposition confirms that essential positions are indeed sufficient to describe the full structure of a term graph (up to isomorphism):

▶ **Proposition 43** ([4]). *Given two term graphs $g, h \in \mathcal{G}^\infty(\Sigma)$, there is a Δ-homomorphism $\phi\colon g \to_\Delta h$ iff, for all $\pi, \pi' \in \mathcal{P}^e(g)$, we have*

$$\text{(a) } \pi \sim_g \pi' \implies \pi \sim_h \pi', \text{ and } \quad \text{(b) } g(\pi) = h(\pi) \quad \text{whenever} \quad g(\pi) \notin \Delta.$$

▶ **Lemma 44** ([4]). *Let $S = (g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\lambda}$ be an open reduction p-converging to g_λ.*

(i) *If there is some $\alpha < \lambda$ such that $\pi \in \mathcal{P}(g_\alpha)$ and, for all $\alpha \leq \iota < \lambda$, π does not pass through n_ι in g_ι, then $g_\iota(\pi) = g_\alpha(\pi)$ for all $\alpha \leq \iota \leq \lambda$.*

(ii) *If $\pi \in \mathcal{P}_\not\perp(g_\lambda)$, then there is some $\alpha < \lambda$ such that, for all $\alpha \leq \iota < \lambda$, $g_\iota(\pi) = g_\lambda(\pi)$ and π does not pass through n_ι in g_ι.*

▶ **Proposition 45** ([4]). *A term graph $g \in \mathcal{G}^\infty(\Sigma)$ is finite iff $\mathcal{P}^e(g)$ is finite.*

▶ **Lemma 46** ([4]). *Let $S = (g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\lambda}$ be an open reduction p-converging to g_λ and $\pi_1 \sim_{g_\lambda} \pi_2$. Then there is some $\alpha < \lambda$ such that $\pi_1 \sim_{g_\iota} \pi_2$ for all $\alpha \leq \iota < \lambda$.*

A.2 Residuals

We first mention the key properties from Barendregt et al. [7] that motivate the definition of residuals and projections:

▶ **Proposition 47** (pre-reduction step residuals, [7]). *Let (n, ρ) and (n', ρ') be disjoint redex occurrences in a term graph g, with matching $\mathcal{V}$-homomorphisms ϕ and ϕ', respectively, and let $g \mapsto_{n,\rho} h$. Then n' is not a node in h, or there is a redex occurrence (n', ρ') in h.*

▶ **Proposition 48** (reduction step projections, [7]). *Given two reduction steps $\psi\colon g \to h$ and $\psi'\colon g \to h'$ contracting two weakly disjoint redex occurrences, there are two reductions $\psi'/\psi\colon h \to^{\leq 1} g'$ and $\psi/\psi'\colon h' \to^{\leq 1} g'$.*

The following proposition confirms the claim that, for m-convergent reductions, the definition can be simplified by omitting the requirement that residual positions have to be in the set of non-$\bot$ positions of the final term graph.

▶ **Proposition 49.** *Let $\mathcal{R}$ be a weakly non-overlapping GRS $\mathcal{R}$, $S\colon g_0 \overset{m}{\twoheadrightarrow}^\alpha_\mathcal{R} g_\alpha$ open, and (n, ρ) a redex occurrence in g_0 with ρ a rule in $\mathcal{R}$. Then $\liminf_{\iota\to\alpha}(n, \rho)//S|_\iota \subseteq \mathcal{P}_\not\perp(g_\alpha)$.*

Proof. Let $n_\iota = (n, \rho)//S|_\iota$ for each $\iota < \alpha$. To prove that $\liminf_{\iota\to\alpha} n_\iota \subseteq \mathcal{P}_\not\perp(g_\alpha)$, we assume some $\pi \in \liminf_{\iota\to\alpha} n_\iota$ and show that $\pi \in \mathcal{P}(g_\alpha)$. Then $\pi \in \mathcal{P}_\not\perp(g_\alpha)$ follows as g_α is total. Since $\pi \in \liminf_{\iota\to\alpha} n_\iota$, there is some $\beta < \alpha$ such that $\pi \in n_\iota$ for all $\beta \leq \iota < \alpha$. According to Proposition 51, each n_ι is a node in g_ι, and, therefore, we have that $\pi \in \mathcal{P}(g_\iota)$ for all $\beta \leq \iota < \alpha$. According to Theorem 1, this means that $\pi \in \mathcal{P}(g_\alpha)$. ◀

▶ **Lemma 50.** *Let $\mathcal{R}$ be a weakly non-overlapping GRS $\mathcal{R}$, $S\colon g_0 \overset{m}{\twoheadrightarrow}_\mathcal{R} g_\alpha$, and (n, ρ) a redex occurrence in g_0 with ρ a rule in $\mathcal{R}$. If $(n, \rho)//T = \emptyset$ for some prefix T of S, then $(n, \rho)//S = \emptyset$.*

Proof. Let $\alpha = |S|$ and $\beta = |T|$, i.e. $T = S|_\beta$. We show by induction on $\gamma \leq \alpha$ that $(n, \rho)//S|_\gamma = \emptyset$ if $\beta \leq \gamma$.

The case $\gamma \leq \beta$ is trivial. Let $\gamma = \gamma' + 1 > \beta$. Since $\gamma' \geq \beta$, we obtain by induction hypothesis that $(n, \rho)//S|_{\gamma'} = \emptyset$. Hence, $(n, \rho)//S|_\gamma = \emptyset$, too. Let $\gamma > \beta$ be a limit ordinal. According to the induction hypothesis, we know that $(n, \rho)//S|_\iota = \emptyset$ for all $\beta \leq \iota < \gamma$. Hence, $(n, \rho)//S|_\gamma = \emptyset$, too. ◀

The following proposition confirms the that our generalisation of projections to reductions of arbitrary length is well-defined:

▶ **Proposition 51.** *Let $\mathcal{R}$ be a weakly non-overlapping GRS $\mathcal{R}$, $S\colon g_0 \overset{p}{\twoheadrightarrow}_\mathcal{R} g_\alpha$, and (n, ρ) a redex occurrence in g_0 with ρ a rule in $\mathcal{R}$. If $(n, \rho)//S = m$ is non-empty, then (m, ρ) is a redex occurrence in g_α.*

Proof. Let $S = (g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\alpha}$, and let $c_\iota = g_\iota \backslash n_\iota$ be the reduction context for each step at $\iota < \alpha$. We proceed by an induction on α.

The case $\alpha = 0$ is trivial. If $\alpha = \beta + 1$, then the statement follows from the induction hypothesis according to Proposition 47.

Let α be a limit ordinal, and let $m_\iota = (n, \rho)//S|_\iota$ for all $\iota < \alpha$.

Since $m \neq \emptyset$, we know, by Lemma 50, that $m_\iota \neq \emptyset$ for all $\iota < \alpha$. Hence, we may invoke the induction hypothesis to obtain that (m_ι, ρ) is a redex occurrence in g_ι for each $\iota < \alpha$, which means that we have matching $\mathcal{V}$-homomorphisms $\phi_\iota \colon \rho_l \to_\mathcal{V} g_\iota|_{m_\iota}$ for all $\iota < \alpha$.

By definition, m is a set of positions in g_α, but we have to show that m is also a node in g_α. If $\pi_1, \pi_2 \in m$, then there is some $\beta < \alpha$ such that $\pi_1, \pi_2 \in m_\iota$ for all $\beta \leq \iota < \alpha$. Since m_ι is a node in g_ι, we thus have $\pi_1 \sim_{g_\iota} \pi_2$ for all $\beta \leq \iota < \alpha$. Moreover, $\pi_1, \pi_2 \in m$ implies that $\pi_1, \pi_2 \in \mathcal{P}(g_\alpha)$, which means, according to Theorem 3, that we can choose β large enough such that $\pi_1, \pi_2 \in \mathcal{P}(c_\iota)$ for all $\beta \leq \iota < \alpha$. By Lemma 8, this means that, $\pi_1 \sim_{g_\iota} \pi_2$ implies $\pi_1 \sim_{c_\iota} \pi_2$ for all $\beta \leq \iota < \alpha$. Consequently, by Theorem 3, we have that $\pi_1 \sim_{g_\alpha} \pi_2$. Hence, all positions in m are in the same $\sim_{g_\alpha}$-equivalence class, which means that there is some node m' in g_α with $m \subseteq m'$.

Before we show the converse inclusion, we choose some $\pi^* \in m$. By the inclusion $m \subseteq m'$ proved above, we know that then $\pi^* \in m'$ as well. Moreover, there is some $\beta < \alpha$ such that $\pi^* \in m_\iota$ for all $\beta \leq \iota < \alpha$. We assume some $\pi \in m'$ and show that then $\pi \in m$. Since $\pi^*, \pi \in m'$, we know that $\pi^* \sim_{g_\alpha} \pi$, which means, by Theorem 3, that we can choose β large enough such that $\pi^* \sim_{c_\iota} \pi$ for all $\beta \leq \iota < \alpha$. According to Lemma 8, we thus have that $\pi^* \sim_{g_\iota} \pi$ for all $\beta \leq \iota < \alpha$. Since we know that $\pi^* \in m_\iota$, we can conclude that also $\pi \in m_\iota$ for all $\beta \leq \iota < \alpha$. We then have $\pi \in m$ because the requirement that $\pi \in \mathcal{P}_\not\perp(g_\alpha)$ follows from $\pi^* \sim_{g_\alpha} \pi$ and $\pi^* \in \mathcal{P}_\not\perp(g_\alpha)$.

By combining both inclusions, we obtain that $m = m'$, i.e. m is a node in g_α.

Before we continue, we shall prove an auxiliary claim. To this end, we pick some $\pi^* \in m$. According to the definition of residuals, we then have that $\pi^* \in \mathcal{P}_\not\perp(g_\alpha)$ and that there is some $\beta < \alpha$ with $\pi^* \in m_\iota$ for all $\beta \leq \iota < \alpha$. By Theorem 3, the former implies that we can chose β large enough such that $\pi^* \in \mathcal{P}_\not\perp(c_\iota)$ for all $\beta \leq \iota < \alpha$. By Lemma 8, this means that $c_\iota(\pi^*) = g_\iota(\pi^*)$ and that $\pi^* \notin n_\iota$ for all $\beta \leq \iota < \alpha$. Note that the latter means that $n_\iota \neq m_\iota$ for all $\beta \leq \iota < \alpha$. Since $\mathcal{R}$ is weakly non-overlapping, this implies that the redex occurrences at n_ι and m_ι must be disjoint for all $\beta \leq \iota < \alpha$.

We now proceed to prove the following claim for all $\pi \in \mathcal{P}(\rho_l)$:

$$\pi^* \cdot \pi \in \mathcal{P}(g_\alpha) \text{ and if } \rho_l(\pi) \notin \mathcal{V}, \text{ then } g_\alpha(\pi^* \cdot \pi) = \rho_l(\pi). \tag{1}$$

We prove this claim by induction on the length of π.

If $\pi = \langle\rangle$, then we know, according to the definition of term graph rules, that $\rho_l(\pi) \notin \mathcal{V}$. Hence, using Lemma 41, we can deduce from the matching $\mathcal{V}$-homomorphisms $\phi_\iota \colon \rho_l \to_\mathcal{V} g_\iota|_{m_\iota}$ that $g_\iota|_{m_\iota}(\pi) = \rho_l(\pi)$ for all $\iota < \alpha$. This means that $g_\iota(\pi^* \cdot \pi) = \rho_l(\pi)$ for all $\beta \leq \iota < \alpha$. Moreover, since $\pi^* \cdot \pi = \pi^*$ and $c_\iota(\pi^*) = g_\iota(\pi^*)$, we know that $\pi^* \cdot \pi \in \mathcal{P}(g_\alpha)$ and that $c_\iota(\pi^* \cdot \pi) = \rho_l(\pi)$ for all $\beta \leq \iota < \alpha$. Consequently, by Theorem 3, we have that $g_\alpha(\pi^* \cdot \pi) = \rho_l(\pi)$.

If $\pi = \pi' \cdot \langle i \rangle$, then we know that $\rho_l(\pi')$ is not a nullary symbol and, thus, not in $\mathcal{V}$. By applying the induction hypothesis, we then obtain that $\pi^* \cdot \pi' \in \mathcal{P}(g_\alpha)$ and that $g_\alpha(\pi^* \cdot \pi') = \rho_l(\pi')$. Taken together these two facts imply that $\pi^* \cdot \pi \in \mathcal{P}(g_\alpha)$. If $\rho_l(\pi) \notin \mathcal{V}$, then we may apply Lemma 41, to obtain from the matching $\mathcal{V}$-homomorphisms $\phi_\iota \colon \rho_l \to_\mathcal{V} g_\iota|_{m_\iota}$ that $g_\iota|_{m_\iota}(\pi) = \rho_l(\pi)$ for all $\iota < \alpha$. Since $\pi^* \in m_\iota$ for all $\beta \leq \iota < \alpha$, we thus have that $g_\iota(\pi^* \cdot \pi) = \rho_l(\pi)$ for all $\beta \leq \iota < \alpha$. As we have derived above, the redex occurrences at

m_ι and n_ι are disjoint for all $\beta \leq \iota < \alpha$. Consequently, $\pi^* \cdot \pi$ does not pass through n_ι in g_ι, which according to Lemma 8 implies that $c_\iota(\pi^* \cdot \pi) = g_\iota(\pi^* \cdot \pi)$ for all $\beta \leq \iota < \alpha$. The resulting equality $c_\iota(\pi^* \cdot \pi) = \rho_l(\pi)$ for all $\beta \leq \iota < \alpha$ together with the fact that $\pi^* \cdot \pi \in \mathcal{P}(g_\alpha)$ yields, by Theorem 3, that $g_\alpha(\pi^* \cdot \pi) = \rho_l(\pi)$. That concludes the proof of (1).

Finally, we show that $g_\alpha|_m$ is a ρ-redex. To this end we show the existence of a $\mathcal{V}$-homomorphisms $\phi\colon \rho_l \to_\mathcal{V} g_\alpha|_m$ using Lemma 41.

(a) Let $\pi_1 \sim_{\rho_l} \pi_2$. For each $\iota < \alpha$, the matching $\mathcal{V}$-homomorphism $\phi\colon \rho_l \to_\mathcal{V} g_\iota|_{m_\iota}$ yields, according to Lemma 41, that $\pi_1 \sim_{g_\iota|_{m_\iota}} \pi_2$. Consequently, $\pi^* \cdot \pi_1 \sim_{g_\iota} \pi^* \cdot \pi_2$ for all $\beta \leq \iota < \alpha$. Since $\pi^* \cdot \pi_1, \pi^* \cdot \pi_2 \in \mathcal{P}(g_\alpha)$ by (1), there is, according to Theorem 3, some $\beta \leq \beta' < \alpha$ such that $\pi^* \cdot \pi_1, \pi^* \cdot \pi_2 \in \mathcal{P}(c_\iota)$ for all $\beta' \leq \iota < \alpha$. Hence, by Lemma 8, $\pi^* \cdot \pi_1 \sim_{g_\iota} \pi^* \cdot \pi_2$ implies $\pi^* \cdot \pi_1 \sim_{c_\iota} \pi^* \cdot \pi_2$ for all $\beta' \leq \iota < \alpha$. Again using the fact that $\pi^* \cdot \pi_1, \pi^* \cdot \pi_2 \in \mathcal{P}(g_\alpha)$, we can apply Theorem 3 to obtain that $\pi^* \cdot \pi_1 \sim_{g_\alpha} \pi^* \cdot \pi_2$. Therefore, $\pi_1 \sim_{g_\alpha|_m} \pi_2$ as $\pi^* \in m$.

(b) Let $\rho_l(\pi) \notin \mathcal{V}$. According to (1), we then have that $g_\alpha(\pi^* \cdot \pi) = \rho_l(\pi)$. Since $\pi^* \in m$, we thus have that $g_\alpha|_m(\pi) = \rho_l(\pi)$. ◀

A.3 Compression Property

In this section, we give the missing proofs for the auxiliary lemmas used to prove the compression property.

▶ **Lemma 22.** *Given an open reduction $S = (g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\lambda}$ p-converging to g_λ and a redex occurrence (π, ρ) in g_λ with ρ left-finite, there is a position $\pi \in \mathcal{P}(g_\lambda)$ and some $\alpha < \lambda$ such that (π, ρ) is a redex occurrence in g_ι, and π does not pass through n_ι in g_ι for any $\alpha \leq \iota < \lambda$.*

Proof of Lemma 22. Since (π, ρ) is a redex occurrence in g_λ, there is a matching $\mathcal{V}$-homomorphism $\phi\colon \rho_l \to_\mathcal{V} g_\lambda|_\pi$. By Proposition 43, this means that, for all $\pi_1, \pi_2 \in \mathcal{P}^e(\rho_l)$, we have

$$\pi_1 \sim_{\rho_l} \pi_2 \quad \Longrightarrow \quad \begin{array}{l} \pi \cdot \pi_1 \sim_{g_\lambda} \pi \cdot \pi_2, \text{ and} \\ \rho_l(\pi_1) = g_\lambda(\pi \cdot \pi_1) \quad \text{whenever} \quad \rho_l(\pi_1) \notin \mathcal{V}. \end{array} \tag{2}$$

By definition of term graph rules, we know that $\rho_l(\langle\rangle) \notin \mathcal{V}$. Hence, $g_\lambda(\pi) = \rho_l(\langle\rangle)$ and therefore $g_\lambda(\pi) \neq \bot$. Hence, we may apply Lemma 44(ii) to obtain some $\alpha < \lambda$ such that $\pi \in \mathcal{P}(g_\iota)$ and π does not pass through n_ι in g_ι for all $\alpha \leq \iota < \lambda$. It remains to be shown that there is some $\alpha \leq \gamma < \lambda$ such that (π, ρ) is a redex occurrence in g_ι for all $\gamma \leq \iota < \lambda$.

Since ρ is left-finite, ρ_l is finite, which means, by Proposition 45, that $\mathcal{P}^e(\rho_l)$ is finite. Consequently, the set $P = \{\pi \cdot \pi' \mid \pi' \in \mathcal{P}^e(\rho_l)\}$ is finite as well. Hence, we may repeatedly apply Lemma 46 (once for each pair $\pi \cdot \pi_1, \pi \cdot \pi_2 \in P$) to obtain some $\alpha \leq \beta < \lambda$ such that

$$\pi \cdot \pi_1 \sim_{g_\lambda} \pi \cdot \pi_2 \text{ implies } \pi \cdot \pi_1 \sim_{g_\iota} \pi \cdot \pi_2 \text{ for all } \pi_1, \pi_2 \in \mathcal{P}^e(\rho_l) \text{ and } \beta \leq \iota < \lambda \tag{3}$$

Likewise, we may repeatedly apply Lemma 44(ii) to obtain some $\beta \leq \gamma < \lambda$ such that

$$g_\lambda(\pi \cdot \pi_1) = g_\iota(\pi \cdot \pi_1) \text{ for all } \pi_1 \in \mathcal{P}^e(\rho_l) \text{ with } \rho_l(\pi_1) \notin \mathcal{V} \text{ and } \gamma \leq \iota < \lambda \tag{4}$$

Note that we may use Lemma 44(ii) since rules do not contain $\bot$ and by (2) above we know that $g_\lambda(\pi \cdot \pi_1) = \rho_l(\pi_1)$ for all $\pi_1 \in \mathcal{P}^e(\rho_l)$ with $\rho_l(\pi_1) \notin \mathcal{V}$.

Using both (3) and (4), we can derive from (2), that for all $\pi_1, \pi_2 \in \mathcal{P}^e(\rho_l)$ and $\gamma \leq \iota < \lambda$

$$\pi_1 \sim_{\rho_l} \pi_2 \quad \Longrightarrow \quad \begin{array}{l} \pi \cdot \pi_1 \sim_{g_\iota} \pi \cdot \pi_2, \text{ and} \\ \rho_l(\pi_1) = g_\iota(\pi \cdot \pi_1) \quad \text{whenever} \quad \rho_l(\pi_1) \notin \mathcal{V}. \end{array}$$

By Proposition 43, the above finding implies the existence of a $\mathcal{V}$-homomorphism $\phi_\iota\colon\ \rho_l \to_\mathcal{V} g_\iota|_\pi$ for all $\gamma \leq \iota < \lambda$, i.e. (π, ρ) is a redex occurrence in g_ι. ◀

▶ **Proposition 24** (compression of closed transfinite reductions)**.** *Let $S\colon\ g \xrightarrow{p}\!\!\!\!\rightarrow_\mathcal{R} h$ in a weakly non-overlapping, left-finite GRS $\mathcal{R}$. Then there is a reduction $T\colon\ g \xrightarrow{p}\!\!\!\!\rightarrow_\mathcal{R} h$ that is finite or open but not longer than S.*

Proof of Proposition 24. We proceed by induction on the length of S. The only non-trivial case is where $|S|$ is a successor ordinal greater than ω. That is, $S = S' \cdot \langle\phi\rangle$ with $S'\colon\ g \xrightarrow{p}\!\!\!\!\rightarrow g'$ and $\phi\colon\ g' \to h$. By induction hypothesis there is a $T'\colon\ g \xrightarrow{p}\!\!\!\!\rightarrow g'$ of length at most $|S'|$. If T' is finite, then so is $T' \cdot \langle\phi\rangle\colon\ g \xrightarrow{p}\!\!\!\!\rightarrow h$. Otherwise, T' is an open reduction. Let (π, ρ) be the redex occurrence contracted in ϕ. We will construct the diagram illustrated in Figure 3.

According to Lemma 22, T' can be factorised into $T_1\colon\ g \xrightarrow{p}\!\!\!\!\rightarrow g_1$ and $T_2\colon\ g_1 \xrightarrow{p}\!\!\!\!\rightarrow g'$ such that (π, ρ) is a redex occurrence in g_1 and π does not pass through the root of any redex contracted in T_2. Consequently, according to Lemma 23, $\pi \in (\pi, \rho)/\!/T_2$, which means that the corresponding projection $\psi/\!/T_2$, where $\psi\colon\ g_1 \to g_2$ contracts the redex occurrence (π, ρ) in g_1, coincides with the single step reduction ϕ. According to Theorem 20, the projection T_2/ψ is of type $g_2 \xrightarrow{p}\!\!\!\!\rightarrow h$. In sum, we have a reduction $\widehat{T} = T_1 \cdot \langle\psi\rangle \cdot T_2/\psi$ with $\widehat{T}\colon\ g \xrightarrow{p}\!\!\!\!\rightarrow h$. Since by construction T_2/ψ is not longer than T_2 and since T_2 is of limit ordinal length, we know that $|\langle\psi\rangle \cdot T_2/\psi| \leq |T_2|$. Consequently, $\left|\widehat{T}\right| < |S|$. Thus, we may apply the induction hypothesis to $\widehat{T}$ to obtain a reduction $T\colon\ g \xrightarrow{p}\!\!\!\!\rightarrow h$ of finite or limit ordinal length. ◀

▶ **Lemma 26.** *For each reduction $S\colon\ g \xrightarrow{p}\!\!\!\!\rightarrow_\mathcal{R} \bot$ in a weakly non-overlapping, left-finite GRS $\mathcal{R}$ with $g \neq \bot$, we find a finite reduction $g \to^*_\mathcal{R} h$ to a redex h.*

Proof of Lemma 26. By Proposition 24, we may assume that S is finite or open. If S is finite, then S is non-empty since $g \neq \bot$. Consequently, we have that $S = T \cdot \langle\phi\rangle$ with ϕ a reduction step contracting a redex at the root. That is, T is a finite reduction from g to a redex. If S is open, we can apply Lemma 15(ii), to obtain that either there is a proper prefix T of S that p-converges to $\bot$, or $\langle\rangle$ is volatile in S. The first case is impossible since $\bot$ is a normal form. In the second case, there is a proper prefix T of S that p-converges to a redex. We show by induction on the length of T, that if T p-converges to a redex, then there is a finite reduction $g \to^*_\mathcal{R} h$ to a redex. By Proposition 24, we may assume that T is finite or open. In the first case, we are done. In the second case, we may apply Lemma 22 to obtain a proper prefix T' of T that p-converges to a redex. We can then apply the induction hypothesis to T' to obtain a finite reduction from g to a redex. ◀

Optimality and the Linear Substitution Calculus*

Pablo Barenbaum[1] and Eduardo Bonelli[2]

1 **Universidad de Buenos Aires, Buenos Aires, Argentinia; and**
Université Paris 7, Paris, France; and
Stevens Institute of Technology, Hoboken, NJ, USA
2 **Universidad Nacional de Quilmes and CONICET, Buenos Aires, Argentinia; and**
Stevens Institute of Technology, Hoboken, NJ, USA

Abstract

We lift the theory of optimal reduction to a decomposition of the lambda calculus known as the *Linear Substitution Calculus* (LSC). LSC decomposes β-reduction into finer steps that manipulate substitutions in two distinctive ways: it uses *context rules* that allow substitutions to act "at a distance" and rewrites modulo a set of *equations* that allow substitutions to "float" in a term. We propose a notion of redex family obtained by adapting Lévy labels to support these two distinctive features. This is followed by a proof of the finite family developments theorem (FFD). We then apply FFD to prove an optimal reduction theorem for LSC. We also apply FFD to deduce additional novel properties of LSC, namely an algorithm for standardisation by selection and normalisation of a linear call-by-need reduction strategy. All results are proved in the axiomatic setting of Glauert and Khashidashvili's *Deterministic Residual Structures*.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Rewriting, Lambda Calculus, Explicit Substitutions, Optimal Reduction

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.9

1 Introduction

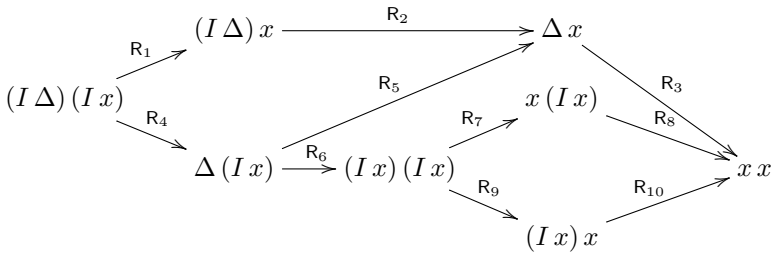
The λ-calculus distills the essence of functional programming languages. Programs are represented as syntactic terms, and execution corresponds to repeated simplification of these terms using a reduction rule called *β-reduction*. The study of the λ-calculus has produced a vast body of work, by no means limited to functional programming. It has also played a key role in laying the foundations of modern *rewriting theory*. Rewriting is an abstract model of computation in which rather than syntactic terms and their step-by-step reduction, one considers sets of *arrows* over arbitrary *objects*. The λ-calculus is an example of a rewriting system, but there are many other ones, such as graph rewriting systems or first-order term rewriting systems. The impact of the λ-calculus in rewriting is that its study has suggested generalizations of numerous properties to abstract rewriting frameworks.

There are many variants of the λ-calculus. In its simplest presentation, it consists of a unique *reduction rule* β that models the application of a function to an argument. Despite the conciseness of its definition, the study of the λ-calculus unveils surprisingly rich mathematical structures. One example is its *denotational semantics*, which attempts to provide *models* for the λ-calculus, and motivates the theory of *domains*. Another example arises from attempting

* Work partially supported by PICT-2012-2747 (Ministerio de Ciencia, Tecnología e Innovación Productiva, Argentina) and LIA INFINIS.

to compare *derivations*. Given that computation is modeled by reduction and that there are multiple ways to reduce a term, how do these choices compare? This requires analyzing the *derivation space* of a term. The derivation space of a term is the set of all derivations, *i.e.* sequence of (composable) β-reduction steps, starting from that term. Establishing whether a particular choice produces derivations that are "better" than others in any reasonable sense involves *comparing* the resulting derivations. This, in turn, involves tracking steps around in order to relate the steps of one derivation to those of another one, hence determining that they *correspond* to each other. *Theories of residuals* attempt to provide a framework for analyzing the derivation space.

An example in the λ-calculus follows in order to provide a better intuition on what is meant by a theory of residuals. Consider the term $(I\,\Delta)\,(I\,x)$, where Δ stands for $\lambda x.x\,x$ and I for $\lambda x.x$. Below we depict the derivation space of this term. As mentioned, a study of the *structure* of this space involves understanding how derivations are related and, since derivations are built from β-steps, how β-steps from one derivation are related to those of another.



An example of a derivation is $\mathsf{R}_4;\mathsf{R}_6;\mathsf{R}_7;\mathsf{R}_8$. It consists of four β-steps denoted R_4, R_6, R_7, and R_8. Notice that the derivation $\mathsf{R}_4;\mathsf{R}_6;\mathsf{R}_7;\mathsf{R}_8$ essentially performs the same steps as the derivation $\mathsf{R}_4;\mathsf{R}_6;\mathsf{R}_9;\mathsf{R}_{10}$ since the derivations $\mathsf{R}_7;\mathsf{R}_8$ and $\mathsf{R}_9;\mathsf{R}_{10}$ do the same computational work, namely they reduce the two copies of $(I\,x)$ in $(I\,x)\,(I\,x)$, only in a different order (reducible subterms such as $(I\,x)$ are called *redexes*). This suggests *algebraic principles* over derivations, such as $\mathsf{R}_7;\mathsf{R}_8 \simeq \mathsf{R}_9;\mathsf{R}_{10}$. Not all steps can be commuted. For example, R_4 cannot be commuted with R_6 because the former *creates* the latter. Note also that, we write $\mathsf{R}_7;\mathsf{R}_8 \simeq \mathsf{R}_9;\mathsf{R}_{10}$ and not $\mathsf{R}_7;\mathsf{R}_8 \simeq \mathsf{R}_8;\mathsf{R}_7$ because R_9 is the form that R_8 adopts when it is fired from $(I\,x)\,(I\,x)$ rather than from $x\,(I\,x)$; we say that R_8 is a *residual* or what is left of R_9 after R_7. As may be gleamed from this preliminary discussion, it soon becomes clear that any prospective algebraic principles must arise from identifying β-steps and tracking them along derivations. Such *theories of residuals* mark and track β-steps. However, this is just the starting point of an analysis of the structure of the derivation space since, when one attempts to prove properties of derivations, one realizes that more general principles are required. The principles include the following, presented in increasing level of complexity:

- *Finite Developments*: marking and tracking *sets* of β-steps in a term and showing that their reduction terminates;
- *Finite Family Developments*: marking and tracking *sets* of β-step that may have been created along the way in a derivation and also showing their termination properties;
- *Redex Families*: identifying created β-steps that are related in the sense that they could be shared;
- *Optimal Reduction*: the apex of residual theory.

Optimal reduction characterizes derivations in the derivation space that are shortest in a precise sense and has close ties with Geometry of Interaction [16]. It arose with a clear motivation in the implementation of the λ-calculus since it addresses the concern of avoiding

unnecessary β-steps. This same motivation, bridging the gap between programming languages and their implementation, is shared by *Calculi with Explicit Substitutions*.

Calculi with Explicit Substitutions (ES). Substitution in the λ-calculus is a nontrivial metalanguage operation that simultaneously replaces every occurrence of a variable by a given term. In contrast, in actual implementations of functional programming languages it usually takes various steps to perform a substitution. For example, variables might be bound to values in an environment, and looked up in the environment whenever needed. Calculi with ES were introduced to bridge the gap between the λ-calculus and its implementations. They are characterized by the presence of an explicit operator in the object language for modeling substitution. A paradigmatic example calculus with ES is $\lambda\sigma$ [1] which includes, among others, rules `beta`[1] $(\lambda x.s)\,t \mapsto s[x/t]$, where $s[x/t]$ denotes an ES, and `app` $(s\,t)[x/u] \mapsto s[x/u]t[x/u]$, for propagating substitutions over applications. Unfortunately, these rules produce a critical pair rendering $\lambda\sigma$ a syntactically *non*-orthogonal system, a situation common to most known calculi with ES, as depicted below where $\to_{\texttt{beta}}$ means application of the `beta`-rule in an arbitrary context:

$$s[x/t][y/u] \;{}_{\texttt{beta}}\!\leftarrow ((\lambda x.s)\,t)[y/u] \to_{\texttt{app}} (\lambda x.s)[y/u]t[y/u]$$

The `beta`-step in the middle term has been "erased" in the right term because `beta` and `app` overlap. It is unclear how to devise a reasonable residual theory in such a situation[2]. The λ-calculus is thus set apart from traditional calculi with ES since in the latter the lack of orthogonality makes it impossible to address a proper theory of residuals, let alone optimality.

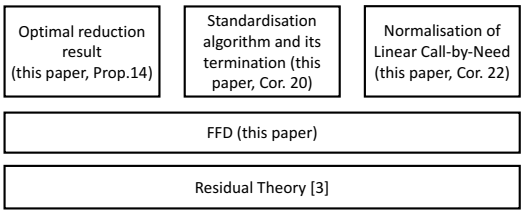
The Linear Substitution Calculus (LSC). The LSC is a calculus with ES introduced rather recently [6]. It is based on a *contextual* approach: rewriting rules are expressed using contexts, which allows for non-local interactions between subterms, and obviates the need to propagate explicit substitutions. It is also equipped with a relation of *structural equivalence* between terms, which reflects the exact correspondence between terms and their encoding as proof nets (which are graphs), linear logic being the domain in which the LSC was originally conceived.

The fact that the LSC encodes a graph-rewriting system based on proof nets, rather than *ad hoc* syntactic machinery for implementing explicit substitutions, is one of the reasons for it being relatively well-behaved. In particular, the LSC does not suffer from the above mentioned problems of other calculi with ES. Recent work has shown that, even though the LSC is not syntactically orthogonal, it enjoys *semantical orthogonality*, which means that it can be given a sensible *theory of residuals* [4]. On the other hand, not all expected properties of residuals that hold for the λ-calculus turn out to hold for LSC (*e.g.* *enclave* and *stability* fail [4]). Besides, the very same fact that the LSC encodes a graph-rewriting system is the source of some technical challenges, especially because the encoding is based on two distinctive features: the use of context rules and a notion of structural equivalence. One complication is that the usual tree-like representation of terms and nesting of redexes that guide our intuition in the λ-calculus and first-order term rewriting no longer applies. *E.g.*, in the term $(x\,x)[x/y]$ either of the two occurrences of x might be replaced by y, so there are two redexes $(x\,x)[x/y] \to (y\,x)[x/y]$ and $(x\,x)[x/y] \to (x\,y)[x/y]$. These redexes overlap in

[1] $\lambda\sigma$ is actually based on de Bruijn indices, we use variable names for expository purposes.
[2] There are some attempts at addressing residual theories for syntactically non-orthogonal systems [14, 23].

the standard tree reading of terms, yet they should by all means be considered "independent" redexes. Another complication is that redex creation may take place at a distance, such as in the step $(x\,y)[x/I] \to (I\,y)[x/I]$, in which the substitution of x by the identity creates a beta-redex. Anyhow, enough properties are satisfied by LSC's residual theory for it to be a reasonable starting point for following the path set by the λ-calculus: finite developments, finite family developments, redex families and optimal reduction.

Goal and value of the paper. This paper attempts to reclaim, for the LSC, the status of the λ-calculus by providing a theory of optimality for it. The key technical result on which it builds is a *Finite Family Developments (FFD)* theorem (Thm. 4 on page 9). FFD is used as a tool to develop various novel results, including optimal reduction itself, termination of standardisation procedures, and normalisation strategies. All results in this paper are proved in an axiomatic setting, namely *Deterministic Residual Structures* [15], whose axioms LSC is shown to comply with.



The reader will no doubt realize the technical nature of this paper. Standardisation, normalisation and, most notably, optimal reduction are known to be technical in themselves. The LSC is not much of an aid in this sense, its use of context rules and rewriting modulo a set of equations only seem to make matters yet more technical. We are well aware of this fact and have strived to present the material in such a way that the reader is able to see through the technicalities and perceive the value of this paper, namely how it manages to lift the theory of optimal reduction to *refinements* of the λ-calculus.

Structure of this paper. Sec. 2 defines LSC. We also review the definition of residuals for LSC and present *Deterministic Residual Structures* [15]. The Lévy labeled LSC is presented in Sec. 3. The *Finite Family Developments Theorem* is addressed in Sec. 4, its proof broken down into three principles. Sec. 5 addresses optimal reduction: we recall the notion of *Deterministic Residual Structure* from [15] and then prove that our labeled LSC is an instance of such structure. Sec. 6 introduces standardisation by selection (of multi-redexes) and proves termination. Sec. 7 studies a linear call-by-need strategy and proves that it normalizes. We conclude in Sec. 8. Proofs of all results are included in the extended version.

Related Work. The literature on FD is quite extensive; the reader is invited to consult [27, Ch. 4]. Some abstract notions of rewriting establish FD as an axiom [25, 21, 15]. For classical references to FFD there is [18, 12]. FFD generalizes Hyland-Wadsworth labels which records the depth of the labels [27, 8.4.4]. Also, it is referred to as *Generalized Finite Developments* in [19]. FFD was extended to higher-order rewriting [28, 26]. LSC was introduced by Milner [24] and then adopted by Accattoli and Kesner [6, 4] although similar ideas were also developed by de Bruijn and Nederpelt (see [8] for additional references). LSC has somewhat revived the explicit substitutions community given its success in explaining results in the classical λ-calculus (*e.g.* cost models, call-by-value solvability, call-by-value on open terms, linear head reduction and abstract machines, etc.) [9, 3, 7, 5, 8]. Regarding

standardisation for LSC, [4] proves the existence and uniqueness of standard derivations. However, standardisation algorithms are not studied. Residuals for calculi with ES have also been studied by Melliès [21, 22] where he developed a general theory of rewriting and applied it, among others, to $\lambda\sigma$ [1]. Regarding labels, ES and sharing there is some work [20, 13], however it all addresses weak reduction. We should also mention [29] which uses a calculus of ES and suggests an optimal reduction result for it. However, no proofs are supplied.

2 The Linear Substitution Calculus

Given *variables* $x, y, z, \ldots$, the set $\mathcal{T}$ of **terms** is defined by the grammar:

$$t, s \quad ::= \quad x \mid t\,s \mid \lambda x.t \mid t[x/s]\,.$$

A term of the form $t[x/s]$ is called a **substitution**. The notion of free and bound variables is defined as usual, in particular $\lambda x.t$ and $t[x/s]$ bind all free occurrences of x in t. We write $\mathsf{fv}(s)$ (resp. $\mathsf{bv}(s)$) for the set of free (resp. bound) variables of s. A **context** is a term with a unique occurrence of a singled-out variable $\Box$ called a hole. If $\mathtt{C}$ is a context, then $\mathtt{C}\langle t\rangle$ is the term resulting from replacing the hole in $\mathtt{C}$ with t (possibly resulting in the capture of free variables of t in $\mathtt{C}$). We write $\mathtt{C}\langle\!\langle t\rangle\!\rangle$ when the free variables of t are not captured by $\mathtt{C}$.

Terms are considered up to a set of **structural equations** that allow commuting some substitutions around, in order to quotient out the order imposed by the fact that terms are trees rather than graphs, and to reflect more closely their correspondence with proof-nets. **Structural equivalence**, written $t \sim s$, is the reflexive, symmetric, transitive, and contextual closure of the following axioms:

$$\begin{array}{rcll}
(\lambda x.t)[y/s] & \sim_{\lambda} & \lambda x.t[y/s] & \text{if } x \neq y \text{ and } x \notin \mathsf{fv}(s) \\
(t\,s)[x/u] & \sim_{@} & t[x/u]\,s & \text{if } x \notin \mathsf{fv}(s) \\
t[x/s][y/u] & \sim_{com} & t[y/u][x/s] & \text{if } x \neq y,\ x \notin \mathsf{fv}(u), \text{ and } y \notin \mathsf{fv}(s)
\end{array}$$

▶ **Definition 1.** The LSC is the pair $\langle \mathcal{T}, \to\rangle$ where $\to$ is defined by the rules $\{\mathsf{db}, \mathsf{ls}\}$ modulo the equations $\{\sim_{\lambda}, \sim_{@}, \sim_{com}\}$, *i.e.* $t \to u$ if and only if $t \sim t'(\to_{\mathsf{db}} \cup \to_{\mathsf{ls}})u' \sim u$. Here $\to_{\mathsf{db}}$ is $\mathtt{C}\langle \mapsto_{\mathsf{db}}\rangle$ (*i.e.* the contextual closure of $\mapsto_{\mathsf{db}}$) and $\to_{\mathsf{ls}}$ is $\mathtt{C}\langle \mapsto_{\mathsf{ls}}\rangle$, $\mapsto_{\mathsf{db}}$ and $\mapsto_{\mathsf{ls}}$ being[3]:

$$(\lambda x.t)\mathtt{L}\ s \mapsto_{\mathsf{db}} t[x/s]\mathtt{L} \qquad\qquad \mathtt{C}\langle\!\langle x\rangle\!\rangle[x/t] \mapsto_{\mathsf{ls}} \mathtt{C}\langle t\rangle[x/t]$$

▶ Remark. Originally LSC includes $\to_{\mathsf{gc}}$, defined as the contextual closure of: $t[x/s] \mapsto_{\mathsf{gc}} t$, if $x \notin \mathsf{fv}(t)$. However, in the literature it is often ignored: dropping it simplifies the metatheory (*e.g.* LSC with $\to_{\mathsf{gc}}$ does not enjoy *stability* [4]; *cf.* Rem. 4) at no loss of generality since $\to_{\mathsf{gc}}$ can be postponed past $\to_{\mathsf{db}}$ and $\to_{\mathsf{ls}}$.

A **LSC-step** $(\mathsf{R}, \mathsf{S}, \ldots)$ is either a pair of the form $\langle \mathtt{C}, (\lambda x.t)\mathtt{L}\ s\rangle$ (a db-step) or a triple of the form $\langle \mathtt{D}, \mathtt{C}\langle\!\langle x\rangle\!\rangle[x/t], \mathtt{C}\rangle$ (an ls-step). Steps, as defined here, are often also called *redexes*. We write $\mathsf{src}(\mathsf{R})$ and $\mathsf{tgt}(\mathsf{R})$ for the source and target of R, respectively. Two redexes are said to be **coinitial** (resp. **cofinal**) if their sources (resp. targets) coincide. A **derivation** $(\rho, \sigma, \ldots)$ is a sequence of steps $\mathsf{R}_1 \ldots \mathsf{R}_n$ s.t. $\mathsf{src}(\mathsf{R}_i) = \mathsf{tgt}(\mathsf{R}_{i-1})$ for $i \in 2..n$. We write ϵ for the empty derivation and $t \twoheadrightarrow s$ if there is a derivation from t to s and say that t is its source and s its target (empty derivations are assumed to be indexed by terms). *E.g.*:

$$\begin{array}{ccccccc}
 & & (\lambda x.\lambda y.xyx)II & \to_{\mathsf{db}} & (\lambda y.xyx)[x/I]I & \to_{\mathsf{ls}} & (\lambda y.Iyx)[x/I]I \\
\to_{\mathsf{db}} & (\lambda y.z[z/y]x)[x/I]I & \sim & & ((\lambda y.z[z/y]x)I)[x/I] & \to_{\mathsf{db}} & (z[z/y]x)[y/I][x/I]
\end{array}$$

[3] For the $\mapsto_{\mathsf{db}}$ rule we have opted to use the more familiar $t\mathtt{L}$ rather than $\mathtt{L}\langle t\rangle$.

Residuals for LSC [4]. Given markers $\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$, **marked terms**[4] are defined as follows, where α ranges over markers: $t, s ::= x \mid x^{\alpha} \mid t\, s \mid \lambda x.t \mid \lambda x^{\alpha}.t \mid t[x/s]$. Since markers are intended to mark redexes, we consider only *well-marked terms*: terms where marks are only placed on redexes (*cf.* [4]). For example, $\lambda x^{\mathbf{a}}.x$ and $\lambda x.x^{\mathbf{a}}$ are not well-marked. **Marked reduction** $\overset{\alpha}{\to}$ on well-marked terms is defined as the contextual closure of the following rewriting rules, where the contexts below are also well-marked:

$$(\lambda x^{\mathbf{a}}.t)\mathtt{L}s \overset{\mathbf{a}}{\mapsto}_{\mathtt{dB}} t[x/s]\mathtt{L} \qquad\qquad \mathtt{C}\langle\!\langle x^{\mathbf{a}} \rangle\!\rangle[x/t] \overset{\mathbf{a}}{\mapsto}_{\mathsf{ls}} \mathtt{C}\langle\!\langle t \rangle\!\rangle[x/t]$$

We write $\mathtt{Red}(s)$ (resp. $\mathtt{Red}_{\mathbf{a}}(s)$) for the set of redexes (resp. marked $\mathbf{a}$) in s. The **set of residuals of** R **after** S is given by $\mathsf{R}/\mathsf{S} := \{\mathtt{Red}_{\mathbf{a}}(u') \mid \mathtt{mark}(t, \mathsf{R}, \mathbf{a}) \overset{\mathsf{S}}{\to} u'\}$, where $\mathtt{mark}(t, \mathsf{R}, \mathbf{a})$ denotes the result of marking redex R in t with $\mathbf{a}$. Given steps S and T such that $\mathsf{tgt}(\mathsf{S}) = \mathsf{src}(\mathsf{T})$, we say that S **creates** T if there is no R such that $\mathsf{R}/\mathsf{S} = \mathsf{T}$. A **multistep** is a non-empty finite set $\mathcal{M}$ of coinitial steps. The residual relation may be extended to multisteps as expected: $\mathcal{M}/\mathsf{S} \overset{\text{def}}{=} \bigcup_{\mathsf{R}\in\mathcal{M}} \mathsf{R}/\mathsf{S}$. Also, we may define the residual of a set of steps after a derivation: $\mathcal{M}/\epsilon \overset{\text{def}}{=} \mathcal{M}$, and $\mathcal{M}/\mathsf{S}\sigma \overset{\text{def}}{=} (\mathcal{M}/\mathsf{S})/\sigma$. Examples of the residual relation follow [4]: let $v = (x^{\mathbf{b}}x^{\mathbf{b}}x^{\mathbf{c}}y^{\mathbf{c}})[x/y][y/w]$, $\mathsf{S} = \langle \Box[y/w], (x^{\mathbf{b}}x^{\mathbf{b}}x^{\mathbf{c}}y^{\mathbf{c}})[x/y], x^{\mathbf{b}}\Box\, x^{\mathbf{c}}y^{\mathbf{c}}\rangle$ (so that $v \overset{\mathsf{S}}{\to} (x^{\mathbf{b}}y\, x^{\mathbf{c}}y^{\mathbf{c}})[x/y][y/w]$), and $\mathsf{R} = \langle \Box[y/w], (x^{\mathbf{b}}x^{\mathbf{b}}x^{\mathbf{c}}y^{\mathbf{c}})[x/y], \Box\, x^{\mathbf{b}}x^{\mathbf{c}}y^{\mathbf{c}}\rangle$. Observe that $\mathtt{mark}(v, \mathsf{R}, \mathbf{a}) = (x^{\mathbf{a}}x^{\mathbf{b}}x^{\mathbf{c}}y^{\mathbf{c}})[x/y][y/w] \overset{\mathsf{S}}{\to} (x^{\mathbf{a}}yx^{\mathbf{c}}x^{\mathbf{c}})[x/y][y/w]$. Therefore, if $\mathcal{M} = \{\mathsf{R}\}$, then $\mathcal{M}/\mathsf{S} = \{\mathsf{R}'\}$ where $\mathsf{R}' = \langle \Box\,[y/w], (x^{\mathbf{b}}y\, x^{\mathbf{c}}y^{\mathbf{c}})[x/y], \Box\, y\, x^{\mathbf{c}}y^{\mathbf{c}}\rangle$. Suppose now that $\mathcal{M} = \mathtt{Red}_{\mathbf{c}}(v)$. Then a similar analysis for each $\mathsf{R} \in \mathcal{M}$ yields $\mathcal{M}/\mathsf{S} = \{\mathsf{R}_1, \mathsf{R}_2\}$ where $\mathsf{R}_1 = \langle \Box\,[y/w], (x^{\mathbf{b}}y\, x^{\mathbf{c}}y^{\mathbf{c}})[x/y], x^{\mathbf{b}}y\,\Box\, y^{\mathbf{c}}\rangle$ and $\mathsf{R}_2 = \langle \Box\,, v, (x^{\mathbf{b}}y\, x^{\mathbf{c}}\Box)[x/y]\rangle$.

▶ Remark. Structural equivalence $\sim$ can be lifted to well-marked terms and the residual relation on steps shown to pass the equations in the sense that they induce a bijection between the steps they relate. Moreover, $\sim$ is a strong bisimulation with respect to $\to$ [4].

Marks are useful to study *developments*. For any $\mathcal{M} \subseteq \mathtt{Red}(t)$, a (possibly infinite) derivation from t, $\rho = \mathsf{R}_1\mathsf{R}_2\ldots$, is a **development** of $\mathcal{M}$ iff $\mathsf{R}_i \in \mathcal{M}/\mathsf{R}_1 \ldots \mathsf{R}_{i-1}$ for all i. A development ρ of $\mathcal{M}$ is said to be **complete** if it is *maximal*, *i.e.* if there is no non-empty derivation σ s.t. $\rho\sigma$ is also a development of $\mathcal{M}$. Note that if ρ is finite, then $\mathcal{M}/\rho = \emptyset$. *E.g.* $t_0 = (xx)[x/t] \to (xt)[x/t] \to (tt)[x/t]$ is a complete development of the set containing the two ls-steps of t_0.

An Abstract Framework: Deterministic Residual Structures

Abstract rewriting frameworks, such as *Orthogonal Axiomatic Rewrite Systems* [21] and *Deterministic Residual Structures (DRS)* [15], single out properties that well-behaved residuals should enjoy. LSC with the above defined notion of residual satisfies the properties of both[5] of these frameworks. Here we briefly describe DRS since they shall be used when we address the applications of FFD to LSC.

An **Abstract Rewrite System** (ARS) is a tuple $\langle Obj, Stp, \mathsf{src}, \mathsf{tgt}\rangle$ of *objects*, *steps* and functions src and tgt that return the source and target, resp., of a step. Moreover, we assume that ARSs are finitely branching, *i.e.* that there is only a finite number of steps having the same object as source.

[4] [4] speaks of *labeled terms*, we use "marked" to stress that they are not to be confused with Lévy labels introduced in Sec. 3.

[5] Although, see comment on enclave and stability in the introduction.

▶ **Definition 2** (Deterministic Residual Structure)**.** A DRS is an ARS endowed with a residual relation $\_/\_$ satisfying the following axioms:

1. Unique ancestor. If $\mathsf{R} \in \mathsf{R}_1/\mathsf{S}$ and $\mathsf{R} \in \mathsf{R}_2/\mathsf{S}$ then $\mathsf{R}_1 = \mathsf{R}_2$.
2. Acyclicity. If $\mathsf{R} \neq \mathsf{S}$ and $\mathsf{R}/\mathsf{S} = \emptyset$ then $\mathsf{S}/\mathsf{R} \neq \emptyset$.
3. Finite Developments (FD). Let ρ, σ be derivations and $\mathcal{M}$ be a set of coinitial steps.
   a. Finite. If ρ a *development* of $\mathcal{M}$, then ρ is finite.
   b. Cofinal. If ρ, σ are *complete developments* of $\mathcal{M}$, then ρ and σ end in the same term.
   c. Equivalent. If ρ, σ are complete developments of $\mathcal{M}$ then they induce the same residual relation, *i.e.* $\mathsf{R}/\rho = \mathsf{R}/\sigma$ for every step R coinitial with ρ.

In the case of LSC: Acyclicity is immediate; Unique ancestor and Finite Developments are proved in [4]. We next introduce some definitions proper to any DRS.

Each multistep determines a "super-step" by taking (any) complete development of that set. Its target is well-defined by axiom Cofinal. A **multistep reduction** D is a sequence of multisteps $\mathcal{M}_1 \dots \mathcal{M}_n$ s.t. $\mathsf{src}(\mathcal{M}_i) = \mathsf{tgt}(\mathcal{M}_{i-1})$ for $i \in 2..n$.

Define $\tau_1 \mathsf{R} \sigma \tau_2 \equiv^1 \tau_1 \mathsf{S} \rho \tau_2$, where σ is a complete development of S/R and ρ is a complete development of R/S. We define **permutation equivalence**, $\equiv$, as the reflexive and transitive closure of $\equiv^1$. Note that $\rho \equiv \sigma$ implies $/\rho = /\sigma$ (*i.e.* they induce the same residual relation).

Let $\mathcal{X}$ be a set of objects in a DRS. An object s is **$\mathcal{X}$-normalizing** if there is a derivation from s to an object in $\mathcal{X}$. We call a step $\mathsf{R} \in \mathtt{Red}(s)$ **$\mathcal{X}$-needed** if at least one residual of it is contracted in any reduction from s to an object in $\mathcal{X}$. *E.g.* for $\mathcal{X}$ the set of normal forms, the underlined step is needed in $\lambda x.\underline{II}$, but not if $\mathcal{X}$ is the set of abstractions.

A **redex with history** in a DRS is a non-empty derivation, usually written $\rho\mathsf{R}$ to single out the last step. We write $\mathsf{Hist}(t) \stackrel{\text{def}}{=} \{\rho\mathsf{R} \mid \mathsf{src}(\rho) = t\}$ for the set of redexes with history whose source is the object t. The **copy relation** between coinitial redexes with history, written $\rho\mathsf{R} \leq \sigma\mathsf{S}$ is defined to hold if and only if there is a derivation τ such that $\rho\tau \equiv \sigma$ and $\mathsf{S} \in \mathsf{R}/\tau$. The reflexive, symmetric and transitive closure of $\leq$, written $\leftrightsquigarrow$, is called the **family relation**. Its equivalence classes are called **redex families**. *E.g.*. if $\rho : \Delta(\underline{II}I) \to \Delta(II)$ and $\sigma : \underline{\Delta(III)} \to (\underline{II}I)(III) \to (II)(III)$, then $\sigma(\overline{II})(III)$ is in the family of $\rho\Delta(\overline{II})$ since $\sigma(\overline{II})(III) \leftrightsquigarrow \rho\Delta(\overline{II})$.

Let $\mathcal{F}$ be a set of redex families $\{\mathsf{Fam}_{\leftrightsquigarrow}(\rho_i)\}_{i\in I}$ such that $\rho_i \in \mathsf{Hist}(t)$, for some fixed t. A **family development** of $\mathcal{F}$ is a pair ${}_\tau|\rho$ where the first component is a "history" $\tau : t \twoheadrightarrow t'$ and the second component is a (possibly infinite) derivation $\rho = \mathsf{R}_1\mathsf{R}_2 \dots$ from t' such that for every index $i \geq 1$, we have $\mathsf{Fam}_{\leftrightsquigarrow}(\tau\mathsf{R}_1 \dots \mathsf{R}_i) \in \mathcal{F}$. Usually the history τ is empty, ρ starts from t, and we identify ${}_\epsilon|\rho$ with ρ. A family development ${}_\tau|\rho$ of $\mathcal{M}$ is said to be **complete** if it is *maximal*, *i.e.* if there is no non-empty derivation σ s.t. ${}_\tau|\rho\sigma$ is also a family development of $\mathcal{F}$.

3 The Labeled LSC

Given *initial labels* $\mathbf{a}, \mathbf{b}, \mathbf{c}, \dots$ including a distinguished one "$\bullet$", we define **labels** $\mathcal{L}$ as:

$$\alpha, \beta ::= \mathbf{a} \mid \lceil\alpha\rceil \mid \lfloor\alpha\rfloor \mid \mathtt{db}(\alpha) \mid \alpha\,\beta\,.$$

We assume juxtaposition $\alpha\,\beta$ to be associative. Labels that are *not* of the form $\alpha\,\beta$ are called **atomic labels**. Labels of the form $\mathtt{db}(\alpha)$ will be used to leave a trace indicating that a db-step was contracted (*cf.* Rem. 3). Similarly, "$\bullet$" will be used to leave a trace indicating the place in which an ls-step was contracted. The remaining labels play a similar rôle to that of Lévy labels for λ-calculus.

The set of **labeled terms** ($\mathcal{T}^\ell$), **labeled contexts** and **labeled substitution contexts** are defined by the following grammar:

$$\begin{array}{rcl} t,s,u,r,q & ::= & x^\alpha \mid \lambda^\alpha x.t \mid @^\alpha(t,s) \mid t[x/s] \\ \mathtt{C} & ::= & \Box \mid \lambda^\alpha x.\mathtt{C} \mid @^\alpha(\mathtt{C},t) \mid @^\alpha(t,\mathtt{C}) \mid \mathtt{C}[x/t] \mid t[x/\mathtt{C}] \\ \mathtt{L} & ::= & \Box \mid \mathtt{L}[x/t] \end{array}$$

Note that substitutions are not labeled. The **external label** of a term t, written $\ell(t)$, is the label decorating the outermost node of t, jumping over substitutions:

$$\begin{array}{rclcrcl} \ell(x^\alpha) & \stackrel{\text{def}}{=} & \alpha & \qquad & \ell(\lambda^\alpha x.t) & \stackrel{\text{def}}{=} & \alpha \\ \ell(@^\alpha(t,s)) & \stackrel{\text{def}}{=} & \alpha & & \ell(t[x/s]) & \stackrel{\text{def}}{=} & \ell(t) \end{array}$$

We also define the following operation $\alpha : t$ for adding a label to an (already labeled) term, jumping over substitutions:

$$\begin{array}{rclcrcl} \alpha : x^\beta & \stackrel{\text{def}}{=} & x^{\alpha\beta} & \qquad & \alpha : (\lambda^\beta x.t) & \stackrel{\text{def}}{=} & \lambda^{\alpha\beta} x.t \\ \alpha : @^\beta(t,s) & \stackrel{\text{def}}{=} & @^{\alpha\beta}(t,s) & & \alpha : (t[x/s]) & \stackrel{\text{def}}{=} & (\alpha : t)[x/s] \end{array}$$

Note that we have $\ell(t\mathtt{L}) = \ell(t)$ and $\alpha : (t\mathtt{L}) = (\alpha : t)\mathtt{L}$.

We shall require one more operation on labels. The **outermost (resp. innermost) atomic label** of a label α is written $\uparrow(\alpha)$ (resp. $\downarrow(\alpha)$) and defined as:

$$\uparrow(\alpha) \stackrel{\text{def}}{=} \begin{cases} \uparrow(\alpha_1) & \text{if } \alpha = \alpha_1\alpha_2 \\ \alpha & \text{otherwise} \end{cases} \qquad \downarrow(\alpha) \stackrel{\text{def}}{=} \begin{cases} \downarrow(\alpha_2) & \text{if } \alpha = \alpha_1\alpha_2 \\ \alpha & \text{otherwise} \end{cases}$$

These functions are well-defined modulo associativity of juxtaposition. We also write $\uparrow(t)$ for $\uparrow(\ell(t))$.

A labeled term is **initially labeled** if all its labels are initial and pairwise distinct. A labeled term $t \in \mathcal{T}^\ell$ is a **variant** of an (unlabeled) term $t_0 \in \mathcal{T}$ if erasing all the labels from t yields t_0. We say that two labeled terms $t, s \in \mathcal{T}^\ell$ are variants of each other if they are variants of the same unlabeled term. Similarly, we may say that two labeled steps (resp. derivations) are variants of an unlabeled step (resp. derivation), or of each other. Sometimes we write t^ℓ to stand for a labeled variant of an unlabeled term t, and similarly for labeled steps and labeled derivations.

Redex names $\mathcal{RN}$ are defined as follows, where α' stands for the sort of atomic labels $\mu,\nu,\xi ::= \mathtt{db}(\alpha) \mid \alpha' \bullet \alpha'$. Note that, although we often identify redex names with the labels that represent them, they should be regarded as being of different sorts.

▶ **Definition 3** (Labeled LSC)**.** LLSC is the pair $\langle \mathcal{T}^\ell, \to_\ell \rangle$, where $\to_\ell \stackrel{\text{def}}{=} \to_{\ell\,\mathsf{db}} \cup \to_{\ell\,\mathsf{ls}}$, and $\to_{\ell\,\mathsf{db}} \stackrel{\text{def}}{=} \mathtt{C}\langle \mapsto_{\mathsf{db}} \rangle$ and $\to_{\ell\,\mathsf{ls}} \stackrel{\text{def}}{=} \mathtt{C}\langle \mapsto_{\mathsf{ls}} \rangle$. Relations $\mapsto_{\mathsf{db}}$ and $\mapsto_{\mathsf{ls}}$ are defined as:

$$\begin{array}{rcl} @^\alpha((\lambda^\beta x.t)\mathtt{L}, s) & \mapsto_{\mathsf{db}} & \alpha\lceil \mathtt{db}(\beta)\rceil : t[x/\lfloor \mathtt{db}(\beta)\rfloor : s]\mathtt{L} \\ \mathtt{C}\langle\!\langle x^\alpha \rangle\!\rangle[x/t] & \mapsto_{\mathsf{ls}} & \mathtt{C}\langle \alpha \bullet : t\rangle[x/t] \end{array}$$

The **name** of the db-step above is $\mathtt{db}(\beta)$ and that of the ls-step is $\downarrow(\alpha) \bullet \uparrow(t)$. We write $t \xrightarrow{\mu}_\ell s$ whenever there is a step $t \to_\ell s$ such that name of the contracted step is μ. An example of a reduction in LLSC follows, it shows how a db redex can create a db-step:

$$\begin{array}{rl} & @^{\mathbf{a}}(@^{\mathbf{b}}(\lambda^{\mathbf{c}} x.\lambda^{\mathbf{d}} y.x^{\mathbf{e}}, z^{\mathbf{f}}), z^{\mathbf{g}}) \\ \xrightarrow{\mathtt{db}(\mathbf{c})}_\ell & @^{\mathbf{a}}((\lambda^{\mathbf{b}\lceil \mathtt{db}(\mathbf{c})\rceil \mathbf{d}} y.x^{\mathbf{e}})[x/z^{\lfloor \mathtt{db}(\mathbf{c})\rfloor \mathbf{f}}], z^{\mathbf{g}}) \\ \xrightarrow{\mathtt{db}(\mathbf{b}\lceil \mathtt{db}(\mathbf{c})\rceil \mathbf{d})}_\ell & x^{\mathbf{a}\lceil \mathtt{db}(\mathbf{b}\lceil \mathtt{db}(\mathbf{c})\rceil \mathbf{d})\rceil}[y/z^{\lfloor \mathtt{db}(\mathbf{b}\lceil \mathtt{db}(\mathbf{c})\rceil \mathbf{d})\rfloor \mathbf{g}}][x/z^{\lfloor \mathtt{db}(\mathbf{c})\rfloor \mathbf{f}}] \end{array}$$

The other two forms of redex creation in LSC are when a db-step creates an ls-step (*e.g.* $(\lambda x.x)y \to_{\mathsf{db}} x[x/y]$) and when an ls-step creates a db-step (*e.g.* $(xy)[x/I] \to_{\mathsf{ls}} (Iy)[x/I]$).

Structural equivalence (Sec. 2) can be lifted to labeled terms as expected. The resulting **labeled structural equivalence**, also called $\sim$, is a strong bisimulation with respect to labeled reduction. LLSC is thus well-defined over $\sim$-equivalence classes. Furthermore, given two equivalent terms $t_1 \sim t_2$ there is a bijection f between the set of steps of t_1 and the set of steps of t_2 such that $\mathsf{tgt}(\mathsf{R}) \sim \mathsf{tgt}(f(\mathsf{R}))$. The resulting system LLSC/ $\sim$ will also enjoy Church-Rosser and Finite Family Developments that LLSC will be shown to enjoy in Sec. 4.

▶ Remark. Labels of the form $\mathtt{db}(\alpha)$ (not present in Lévy labeling for λ-calculus) are included for technical reasons. Consider the following example in which an ls-step creates a db-step:

$$@^{\mathbf{a}}(x^{\mathbf{b}}, t)[x/\lambda^{\mathbf{c}} y.s] \xrightarrow{\mathbf{b}\bullet\mathbf{c}}_{\ell} @^{\mathbf{a}}(\lambda^{\mathbf{b}\bullet\mathbf{c}} y.s, t)[x/\lambda^{\mathbf{c}} y.s]$$

If the name of the db-step at the right hand side was declared to be the label decorating the λ-node, namely $\mathbf{b}\bullet\mathbf{c}$, it would coincide with the name of the ls-step we have just fired.

4 Finite Family Developments

FFD relies on the following properties of LLSC:

Property 1: Labeled reduction ($\to_\ell$) is weak Church–Rosser.

Property 2: Residuals of a step have the same name: $\mathsf{S}' \in \mathsf{S}/\rho$ implies S and S' have the same name in any labeling of any LLSC derivation ρ.

Property 3: Creation implies name contribution: if R creates S then $\mu \overset{\mathsf{Name}}{\hookrightarrow} \nu$, where μ denotes the name of R and ν denotes the name of S. The latter relation is called *name contribution* and is defined as the transitive closure of the following rules:

1. $\mathtt{db}(\beta) \overset{\mathsf{Name}}{\hookrightarrow} \mathtt{db}(\alpha \lceil \mathtt{db}(\beta) \rceil \gamma)$
2. $\mathtt{db}(\beta) \overset{\mathsf{Name}}{\hookrightarrow} \alpha \bullet \lfloor \mathtt{db}(\beta) \rfloor$ where α is any atomic label.
3. $\downarrow(\alpha) \bullet \uparrow(\beta) \overset{\mathsf{Name}}{\hookrightarrow} \mathtt{db}(\alpha \bullet \beta)$

We next set out to prove FFD. Its precise statement is:

▶ **Theorem 4** (FFD). *Let $\mathcal{F}$ be a finite set of redex families in $\mathsf{Hist}(t)$ for some term t.*

1. *(Finite) there is no infinite family development of $\mathcal{F}$;*
2. *(Cofinal) the complete family developments of $\mathcal{F}$ all end in the same term; and*
3. *(Equivalent) any two complete family developments ρ and σ of $\mathcal{F}$ satisfy $\rho \equiv \sigma$,* i.e. *they are permutation equivalent.*

(Finite). Labeled reduction is clearly not SN since it can simulate β-reduction. However, if we restrict redex names to those that verify a *bounded predicate* [18], then we do obtain SN. A predicate on redex names $P : \mathcal{RN} \to \mathsf{Bool}$ is said to be **bounded** if the set $\{h(\mu) \mid P(\mu) \text{ holds}\}$ is bounded, where the *height* $h(\mu)$ *of a redex name* μ is the height of μ interpreted as a label, and the height of a label is defined as follows[6]:

$$h(\mathbf{a}) \overset{\text{def}}{=} 1 \qquad h(\alpha\beta) \overset{\text{def}}{=} \max\{h(\alpha), h(\beta)\} \qquad h(\mathbf{f}(\alpha)) \overset{\text{def}}{=} 1 + h(\alpha) \text{ if } \mathbf{f} \in \{\lceil\cdot\rceil, \lfloor\cdot\rfloor, \mathtt{db}(\cdot)\}$$

We write $\to_\ell^P$ for labeled reduction restricted to contracting steps whose names verify the predicate P. SN for $\to_\ell^P$ relies on the abstract termination result[7]: WCR $\wedge$ WN $\wedge$ Inc $\implies$

[6] This operation is well-defined modulo associativity of juxtaposition.

[7] Due to Klop and Nederpelt (see for instance [27, Theorem 1.2.3 (iii)]).

SN. WCR follows from **Property 1**, and the local confluence diagram for a pair of coinitial steps R, S is closed with their relative residuals, which have the *same name* as their ancestors (**Property 2**). WN is attained by picking, at each step, a *non-duplicating* redex R. This implies that R itself has no residual, every other $\to_\ell^P$-step $\mathsf{S} \neq \mathsf{R}$ has exactly one residual with the same name (**Property 2**) and steps created by R will have height strictly greater than that of R since creation implies name contribution (**Property 3**). Finally, Inc is rather easy given that we have a bound on P.

▶ **Proposition 5** (Bounded reduction is SN). *Let P be a bounded predicate. Then $\to_\ell^P$ is SN.*

We may now conclude with a proof of (Finite): it follows from Prop. 5, the fact that all names in a redex family are identical, and that we only have a finite set of families. The axiom (Cofinal) follows from confluence of LLSC, a consequence of **Property 1**, Finite and Newman's Lemma (WCR+SN ⇒ CR). The axiom (Equivalent) follows from the fact that LLSC enjoys *algebraic confluence*: the confluence diagram for two coinitial derivations ρ and σ can be closed by tiling it with elementary permutation diagrams. This concludes the proof of FFD.

▶ Remark. We end the section with a remark on *stability*, stated as follows. Let $\mathsf{R} \neq \mathsf{S}$ be coinitial steps and let $\mathsf{T}_1, \mathsf{T}_2, \mathsf{T}_3$ be steps such that $\mathsf{T}_3 \in \mathsf{T}_1/(\mathsf{R}/\mathsf{S})$ and $\mathsf{T}_3 \in \mathsf{T}_2/(\mathsf{S}/\mathsf{R})$. Then there exists a step T_0 such that $\mathsf{T}_1 \in \mathsf{T}_0/\mathsf{R}$ and $\mathsf{T}_2 \in \mathsf{T}_0/\mathsf{S}$. Stability is known to fail if $\to_{\mathsf{gc}}$ is added to LSC (indeed, it suffices to consider the two ways in which a gc-step can be created in a term such as $x[y/z][z/t]$). Stability for LSC is an easy consequence[8] of the fact that residuals of steps have the same name as their ancestors (**Property 2**).

5 Optimal Reduction for LSC

An optimal reduction [18, 10] computes a value, assuming it exists, in the least number of steps. More precisely, if $\mathcal{A}$ and $\mathcal{B}$ are ARSs, we say that $\mathcal{B}$ is a **sub-ARS** of $\mathcal{A}$ if (1) they have the same objects, *i.e.* $Obj(\mathcal{A}) = Obj(\mathcal{B})$, (2) all the steps of $\mathcal{B}$ are also in $\mathcal{A}$, *i.e.* $Stp(\mathcal{B}) \subseteq Stp(\mathcal{A})$, and (3) the source (resp. target) of a step in $\mathcal{B}$ coincides with its source (resp. target) in $\mathcal{A}$. A **strategy** in an ARS $\mathcal{A}$ is a sub-ARS of $\mathcal{A}$ having the same set of objects and normal forms (*cf.* [27, Def. 9.1.1]). If $\mathcal{X}$ is a set of objects, a strategy is **$\mathcal{X}$-optimal** if for any object t the length of any reduction from t to an object $s \in \mathcal{X}$ is minimal among all the possible reductions from t to s. Strategies such as call-by-name and call-by-value are not optimal: the former duplicates arguments and the latter evaluates unnecessary arguments. Call-by-need evaluates only arguments that are needed and stores their value for subsequent lookup and is indeed optimal [11]. However, all these are strategies in the ARS of closed λ-terms with *weak* reduction, in the sense that β-steps are not performed under lambdas: the set of normal forms are the abstractions. It is relatively easy to implement call-by-need in this case since it suffices to share *subterms* by labeling them [11]. Optimal reduction for the ARS of λ-terms with *strong* (*i.e.* unrestricted) reduction is more complicated since it involves reducing *under* lambdas: the set of normal forms is the usual set of β-normal forms. As a consequence, it requires sharing *contexts*, which notably complicates its implementation. Here we concentrate on a characterization of which of these steps should be shared, leaving implementation concerns, such as how to share contexts, for future work.

In the case of LSC, $\mathcal{X}$-optimality is not very interesting when $\mathcal{X}$ is the set of normal forms: since LSC has no erasing rules, all steps are trivially $\mathcal{X}$-needed. *E.g.* the db-step in

[8] Also a consequence of LSC being a Deterministic Family Structure (*cf.* Sec. 5 and Lem. 4.1 of [15]).

$x[y/II]$ is needed to get to the normal form $x[y/I[z/I]]$. However, II may be considered junk in that it is the body of a substitution whose target variable y has no occurrence in x. Therefore, we introduce a more refined notion of *result* as a candidate for our set $\mathcal{X}$. We are only interested in steps in a term t that are not junk in the sense that they have residuals in the gc-normal form. Let $\mathsf{nf}_{\mathsf{gc}}(t)$ stand for the gc-normal form of t. Our candidate $\mathcal{X}$ is the set of **reachable normal forms**, defined as $\mathsf{RNF} \stackrel{\text{def}}{=} \{t \mid \mathsf{nf}_{\mathsf{gc}}(t) \text{ is in } \to_{\mathsf{db} \cup \mathsf{ls}}\text{-normal form}\}$. Later we shall see that it has the properties required of a set of results (*cf.* notion of stable set of objects below).

5.1 An Abstract Framework for Optimal Reduction

An abstract framework for obtaining optimal reduction results was developed by Khasidashvili and Glauert [15]. They introduce axioms on DRS that verse over steps, residuals and redex families and show that if they are satisfied, then an optimal reduction result holds. These axioms are collected in a structure called *Deterministic Family Structures (DFS)*:

$$\text{ARS (Sec. 2)} \subseteq \text{DRS (Def. 2)} \subseteq \text{DFS (Def. 6)}.$$

▶ **Definition 6.** A **Deterministic Family Structure** is a triple $\langle R, \simeq, \hookrightarrow \rangle$, where R is a DRS, $\simeq$ is an equivalence relation between coinitial redexes with history whose equivalence classes are called *families*, and $\hookrightarrow$ is a binary relation of *contribution* between coinitial families. The family of a redex with history $\rho\mathsf{R}$ is written $\mathsf{Fam}_{\simeq}(\rho\mathsf{R})$. Two families are coinitial if their representatives are coinitial. Moreover, the following axioms hold:

1. Initial. If R, S are distinct coinitial steps, then $\mathsf{Fam}_{\simeq}(\mathsf{R}) \neq \mathsf{Fam}_{\simeq}(\mathsf{S})$.
2. Copy. $\leq \subseteq \simeq$. Recall that $\leq$ is the copy relation of DRS.
3. Finite Family Developments. Any derivation that contracts redexes of a finite number of families is finite.
4. Creation. If $\rho\mathsf{R}$ is a redex with history and R creates S, then $\mathsf{Fam}_{\simeq}(\rho\mathsf{R}) \hookrightarrow \mathsf{Fam}_{\simeq}(\rho\mathsf{RS})$.
5. Contribution. Given any two coinitial families $\phi_1, \phi_2 \in \mathsf{Hist}(t)/\simeq$, the relation $\phi_1 \hookrightarrow \phi_2$ holds, if and only if, for every redex with history $\sigma\mathsf{S} \in \phi_2$, there is a redex with history $\rho\mathsf{R} \in \phi_1$ such that $\rho\mathsf{R}$ is a prefix of σ (*i.e.* $\sigma = \rho\mathsf{R}\sigma'$).

A **family reduction** in a DFS is a multistep reduction $\mathcal{M}_1 \dots \mathcal{M}_\mathsf{n}$ such that for each $i \in 1..n$ all the steps in $\mathcal{M}_\mathsf{i}$ belong to the same redex family. More precisely, for all $i \in 1..n$ and any two $\mathsf{R}, \mathsf{S} \in \mathcal{M}_\mathsf{i}$, it holds that $\mathcal{M}_1 \dots \mathcal{M}_{\mathsf{i}-1}\mathsf{R} \simeq \mathcal{M}_1 \dots \mathcal{M}_{\mathsf{i}-1}\mathsf{S}$. A family reduction is **complete** if each $\mathcal{M}_\mathsf{i}$ is a maximal set of steps that have $\mathsf{src}(\mathcal{M}_\mathsf{i})$ as source and belong to the same family. Let $\mathcal{X}$ be a set of objects. A family reduction is **$\mathcal{X}$-needed** if each $\mathcal{M}_\mathsf{i}$ contains at least one $\mathcal{X}$-needed step (*cf.* Sec. 2).

For a set $\mathcal{X}$ of objects to be admitted as a set of results it has to satisfy the following property. A set $\mathcal{X}$ of objects is **stable** if: 1) $\mathcal{X}$ is closed under parallel moves, *i.e.* for any $t \notin \mathcal{X}$, any $\rho : t \twoheadrightarrow s \in \mathcal{X}$, and any $\sigma : t \twoheadrightarrow u$ which does not contain objects in $\mathcal{X}$, the final object of ρ/σ is in $\mathcal{X}$; and 2) $\mathcal{X}$ is closed under unneeded expansion, *i.e.* for any $t \xrightarrow{\mathsf{R}} s$ such that $t \notin \mathcal{X}$ and $s \in \mathcal{X}$, the step R is $\mathcal{X}$-needed. The set of LSC-normal forms and the set of abstractions are stable. Less obvious is the fact that RNF is a stable set. This is non-trivial. For item 1) we show that the set RNF is closed under reduction, which entails that it is closed under parallel moves. For item 2) we strengthen the notion of reachable steps to that of *strongly reachable steps* (reachable steps that are minimal w.r.t. the box order introduced in [4] for the purposes of studying standardisation).

▶ **Lemma 7.** *The set of reachable-normal forms* RNF *is stable.*

The main result of this section is the following theorem. It is a corollary of Prop. 9 and of Theorem 5.2 in [15]:

▶ **Theorem 8.** *Let $\mathcal{X}$ be a stable set of terms of* LSC*. Let t be a $\mathcal{X}$-normalising term. Then any $\mathcal{X}$-needed $\mathcal{X}$-normalizing complete family-reduction $\rho : t \twoheadrightarrow t' \in \mathcal{X}$ is $\mathcal{X}$-optimal,* i.e. *it has a minimal number of family-reduction steps.*

5.2 LSC is a Deterministic Family Structure

Given two coinitial redexes with history $\rho\mathsf{R}$, $\sigma\mathsf{S}$, the binary relations of **family equivalence** $\rho\mathsf{R} \overset{\mathsf{Fam}}{\simeq} \sigma\mathsf{S}$ and **family contribution** $\rho\mathsf{R} \overset{\mathsf{Fam}}{\hookrightarrow} \sigma\mathsf{S}$ are defined as follows. Consider labeled variants $\rho^\ell\mathsf{R}^\ell$ and $\sigma^\ell\mathsf{S}^\ell$ of $\rho\mathsf{R}$ and $\sigma\mathsf{S}$ respectively, starting from the same initially labeled term. Let μ be the name of R^ℓ and let ν be the name of S^ℓ. We declare $\rho\mathsf{R} \overset{\mathsf{Fam}}{\simeq} \sigma\mathsf{S}$ to hold if and only if $\mu = \nu$ and $\rho\mathsf{R} \overset{\mathsf{Fam}}{\hookrightarrow} \sigma\mathsf{S}$ to hold if and only if $\mu \overset{\mathsf{Name}}{\hookrightarrow} \nu$. Relation $\overset{\mathsf{Fam}}{\hookrightarrow}$ is also extended to coinitial families, declaring $\phi_1 \overset{\mathsf{Fam}}{\hookrightarrow} \phi_2$ to hold whenever for any $\rho\mathsf{R} \in \phi_1$ and $\sigma\mathsf{S} \in \phi_2$ we have $\rho\mathsf{R} \overset{\mathsf{Fam}}{\hookrightarrow} \sigma\mathsf{S}$. It is straightforward to check that this is well-defined, regardless of the choice of representatives.

▶ **Proposition 9.** $(\text{LSC}, \overset{\mathsf{Fam}}{\simeq}, \overset{\mathsf{Fam}}{\hookrightarrow})$ *is a Deterministic Family Structure.*

The axioms Initial, Copy, and Creation can be checked by exhaustive case analysis. The Finite family developments axiom has already been established in Thm. 4. The Contribution axiom is more demanding and relies on a non-trivial application of FFD.

6 Standardisation by Selection for LSC

We introduce an abstract notion of *uniform multi-selection strategy*, show that repeated application of this strategy terminates using FFD in any DFS, and finally that two permutation equivalent derivations produce the same multiderivation. Then we instantiate our abstract result to LSC, obtaining an algorithm for standardizing LSC derivations by picking multisteps according to a given parametric partial order on its steps.

Uniform Multi-Selection Strategies. A step R **belongs** to a derivation ρ, written $\mathsf{R} \triangleleft \rho$, if and only if $\rho = \rho_1\mathsf{R}'\rho_2$ and $\mathsf{R}' \in \mathsf{R}/\rho_1$. Given a DRS $\mathcal{A}$, we write Stp^+ for the set of multisteps, *i.e.* non-empty finite sets of coinitial steps, and we let D, E, etc. range over multiderivations, *i.e.* derivations in the DRS whose steps are multisteps. A multistep $\mathcal{M}$ belongs to a derivation ρ, written $\mathcal{M} \triangleleft \rho$, if and only if $\mathsf{R} \triangleleft \rho$ for all $\mathsf{R} \in \mathcal{M}$. If $\mathsf{D} = \mathcal{M}_1 \dots \mathcal{M}_\mathsf{n}$ is a multiderivation, we say that a derivation ρ is a complete development of D if $\rho = \rho_1 \dots \rho_\mathsf{n}$, where each ρ_i is a complete development of the multistep $\mathcal{M}_\mathsf{i}$. By FD a complete development always exists and any two complete developments are permutation equivalent. We write $\partial\mathsf{D}$ to stand for some complete development of D, and ρ/D for $\rho/\partial\mathsf{D}$. A **multi-selection strategy** is a function $\mathbb{M}$ that maps every non-empty derivation ρ to a coinitial multistep $\mathcal{M} \in Stp^+$ such that $\mathcal{M} \triangleleft \rho$ and $\mathcal{M}/\rho = \emptyset$, *i.e.* residuals of every step appear somewhere in the sequence, and there are no residuals of any step left after the sequence. It is, moreover, **uniform** if $\rho \equiv \sigma$ implies $\mathbb{M}(\rho) = \mathbb{M}(\sigma)$ for any non-empty ρ, σ. *E.g.* $\mathbb{M}_{\text{Triv}}(\mathsf{R}\rho) \overset{\text{def}}{=} \{\mathsf{R}\}$ is a (trivial) multi-selection strategy, which is not uniform.

The **multiderivation induced by a multi-selection strategy $\mathbb{M}$ on a derivation** ρ, written $\mathbb{M}^\star(\rho)$, is a sequence of multisteps defined as follows:

$$\mathbb{M}^\star(\epsilon) = \epsilon \qquad \mathbb{M}^\star(\rho) = \mathbb{M}(\rho)\,\mathbb{M}^\star(\rho/\mathbb{M}(\rho)) \quad \text{if } \rho \neq \epsilon$$

Successively applying a multi-selection strategy $\mathbb{M}$ to build a reduction sequence $\mathbb{M}^\star(\rho)$ terminates, as long as the input ρ is finite, *i.e.* recursion is well-founded. This relies on FFD.

▶ **Lemma 10** (Induced multiderivations preserve finiteness)**.** *Suppose that $\mathbb{M}$ is a multi-selection strategy in a DFS. If ρ is finite, then $\mathbb{M}^\star(\rho)$ is also finite.*

By definition, a uniform multi-selection strategy $\mathbb{M}$, when given two permutation equivalent derivations, always selects the same multistep. It, in fact, yields the same multiderivation.

▶ **Lemma 11.** *Let $\mathbb{M}$ be a uniform multi-selection strategy in a DFS, and let ρ, σ be finite derivations. If $\rho \equiv \sigma$ then $\mathbb{M}^\star(\rho) = \mathbb{M}^\star(\sigma)$.*

▶ **Lemma 12.** *Let $\mathbb{M}$ be a multi-selection strategy in a DFS, and ρ a finite derivation. Then $\rho \equiv \partial\mathbb{M}^\star(\rho)$.*

A multiderivation D is said to be **$\mathbb{M}$-compliant** if and only if $\mathbb{M}^\star(\partial\mathsf{D}) = \mathsf{D}$.

▶ **Proposition 13.** *Let $\mathbb{M}$ be a uniform multi-selection strategy in a DFS. For any finite derivation ρ there exists a unique multiderivation D such that $\rho \equiv \partial\mathsf{D}$ and D is $\mathbb{M}$-compliant. Namely, $\mathsf{D} = \mathbb{M}^\star(\rho)$.*

Standardisation Algorithm for LSC. For each term t let $\mathsf{Out}(t)$ be the set of steps whose source is t in LSC, and let $<_t$ be an arbitrary strict *partial order* on $\mathsf{Out}(t)$. The **arbitrary selector** $\mathbb{M}_<$ is defined as follows: $\mathbb{M}_<(\rho) \stackrel{\text{def}}{=} \{\mathsf{R} \mid \mathsf{R}/\rho = \emptyset \text{ and } \mathsf{R} \text{ is minimal}\}$. By minimal we mean that there is no step R' such that $\mathsf{R}'/\rho = \emptyset$ and $\mathsf{R}' <_{\mathsf{src}(\rho)} \mathsf{R}$. Note that $\mathbb{M}_<$ is a non-empty finite set, given that the set $\{\mathsf{R} \mid \mathsf{R}/\rho = \emptyset\}$ is non-empty and finite, so it has at least one minimal element.

▶ **Lemma 14.** *$\mathbb{M}_<$ is a uniform multi-selection strategy.*

▶ **Corollary 15** (Standardisation by arbitrary selection for LSC)**.** *For each finite sequence ρ in* LSC*, there is a unique multiderivation D such that $\rho \equiv \partial\mathsf{D}$ and D is $\mathbb{M}_<$-compliant. Moreover, if the order $<_t$ is computable, then D is computable from ρ, namely $\mathsf{D} = \mathbb{M}^\star_<(\rho)$.*

For example, let $\rho : x[x/t] \to x[x/t'] \to t'[x/t'] \to t''[x/t']$, where $t \to t' \to t''$.

1. If $<^1$ is the trivial partial order in which every step is incomparable, *i.e.* $\mathsf{R} <^1_t \mathsf{S}$ never holds, then $\mathbb{M}^\star_{<^1}(\rho) : x[x/t] \multimap t'[x/t'] \to t''[x/t']$. The first step is a proper multistep.
2. Let $<^2$ be the total left-to-right order, defined so that $\mathsf{R} <^2_t \mathsf{S}$ holds whenever R is to the left of S. Then $\mathbb{M}^\star_{<^2}(\rho) : x[x/t] \to t[x/t] \to t'[x/t] \to t'[x/t'] \to t''[x/t']$.
3. If $<^3$ is the total right-to-left order, defined so that $\mathsf{R} <^3_t \mathsf{S}$ holds if R is to the right of S. Then $\mathbb{M}^\star_{<^3}(\rho) = \rho : x[x/t] \to x[x/t'] \to t'[x/t'] \to t''[x/t']$.

7 Normalisation of the Linear Needed Strategy in LSC

Recall that a **strategy** in an ARS is a sub-ARS having the same objects and normal forms. We write $\mathsf{NF}(\mathcal{A})$ for the set of normal forms of an ARS $\mathcal{A}$, and $t \to_{\mathcal{A}} s$ to emphasize that a given step is in $\mathcal{A}$. If $\mathcal{X}$ is a superset of the normal forms of $\mathcal{A}$, a strategy $\mathbb{S}$ is said to be **$\mathcal{X}$-normalizing** if for every object t such that there exists a reduction $t \twoheadrightarrow_{\mathcal{A}} s \in \mathcal{X}$, every maximal reduction from t in the strategy $\mathbb{S}$ contains an object in $\mathcal{X}$. A sub-ARS $\mathcal{B}$ is **residual-invariant** if for any steps R and S such that $\mathsf{R} \in \mathcal{B}$ and $\mathsf{S} \neq \mathsf{R}$, there exists a step $\mathsf{R}' \in \mathbb{S}$ such that $\mathsf{R}' \in \mathsf{R}/\mathsf{S}$. A sub-ARS $\mathcal{B}$ is **closed** if the set $\mathsf{NF}(\mathcal{B})$ is closed by reduction, *i.e.* $t \to_{\mathcal{A}} s$ and $t \in \mathsf{NF}(\mathcal{B})$ imply $s \in \mathsf{NF}(\mathcal{B})$. Observe that any sub-ARS $\mathcal{B}$ can

be extended to a strategy $\mathbb{S}_{\mathcal{B}}$ by adjoining the steps going out from normal forms, *i.e.* by setting $Stp(\mathbb{S}_{\mathcal{B}}) := Stp(\mathcal{B}) \cup \{\mathsf{R} \in Stp(\mathcal{A}) \mid \mathsf{src}(\mathsf{R}) \in \mathsf{NF}(\mathcal{B})\}$. We will instantiate the following normalisation result to the linear call-by-need strategy of LSC which we define below.

▶ **Proposition 16.** *Let $\mathcal{B}$ be a closed residual-invariant sub-ARS in a DFS. Then the corresponding strategy $\mathbb{S}_{\mathcal{B}}$ is $\mathsf{NF}(\mathcal{B})$-normalizing.*

Needed linear reduction for LSC is the sub-ARS $\mathbb{NL}$ of LSC defined as follows. Need contexts are defined by the grammar $\mathtt{N} ::= \Box \mid \mathtt{N}\,t \mid \mathtt{N}[x/t] \mid \mathtt{N}\langle\langle x \rangle\rangle[x/\mathtt{N}]$. The reduction rule $\to_{\mathbb{NL}}$ is the union of the usual db rule, and the lsnl rule $\mathtt{N}\langle\langle x \rangle\rangle[x/\mathtt{vL}] \mapsto_{\mathsf{lsnl}} \mathtt{N}\langle \mathtt{vL} \rangle[x/\mathtt{vL}]$, both closed by need contexts, where $\mathtt{v}$ stands for a *value*, *i.e.* a term of the form $\lambda y.t$. Note that it is in fact a sub-ARS for LSC, *i.e.* the lsnl rule is a particular case of the ls rule, and closure by need contexts is a particular case of closure by general contexts. A similar, albeit slightly different call-by-need calculus based on LSC has been studied in [2] to relate the execution model of abstract machines with reduction in calculi with ES. In [17] it is shown, via intersection types, that it is a sound and complete implementation of call-by-name.

The set of **needed linear normal forms** NLNF is defined by the grammar $A ::= (\lambda x.t)\mathtt{L} \mid \mathtt{N}\langle\langle x \rangle\rangle$. Terms of the form $(\lambda x.t)\mathtt{L}$ are called **answers**, and $\mathtt{N}\langle\langle x \rangle\rangle$ are called **structures**. In structures, $\mathtt{N}$ does not bind x, the latter called its **needed variable**.

▶ **Corollary 17.** *The strategy $\mathbb{S}_{\mathbb{NL}}$ associated to the sub-ARS $\mathbb{NL}$ is NLNF-normalizing.*

The proof consists in first showing that $\mathsf{NF}(\mathbb{NL})$ coincides with the set NLNF, and then that the sub-ARS $\mathbb{NL}$ is closed residual-invariant. These items rely on a number of lemmata such as the fact that the needed variable in a structure is unique and that answers cannot be written as of the form $\mathtt{N}\langle\Delta\rangle$ where Δ is a redex or a variable not bound by $\mathtt{N}$.

8 Conclusions

The *Linear Substitution Calculus* sits between calculi with ES and the λ-calculus: it has ES but admits a theory of residuals. We devise a theory of optimal reduction for LSC. We start from the theory of residuals developed in [4] and use it to prove a Finite Family Developments result. This is achieved by introducing a Lévy labeling and associated notion of redex family which supports the two distinctive features of LSC, namely its use of *context rules* that allow substitutions to act "at a distance" and also the set of equations modulo which it rewrites which allow substitutions to "float" in a term. We then apply FFD to prove a number of novel results for LSC including: an optimal reduction result, an algorithm for standardisation by selection, and normalization of a linear call-by-need reduction strategy.

Perhaps the most relevant future work is devising an appropriate notion of extraction and showing that all three characterizations (labeling, zig-zag and extraction) of redex family coincide. This is non-trivial and has elided us for some time. Also, there is the topic of graph based implementations, labels and virtual redexes (*cf.* notion of paths in Ch.6 of [10]).

Acknowledgements. To Thibaut Balabonski and Beniamino Accattoli for helpful discussions.

References

1 Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991. doi:10.1017/S0956796800000186.

2 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In J. Jeuring and M. Chakravarty, editors, *Proceedings of ICFP*, pages 363–376. ACM, 2014.

3 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. A strong distillery. In Xinyu Feng and Sungwoo Park, editors, *APLAS 2015, Pohang, South Korea, November 30 – December 2, 2015, Proceedings*, volume 9458 of *LNCS*, pages 231–250. Springer, 2015. doi:10.1007/978-3-319-26529-2_13.

4 Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In Suresh Jagannathan and Peter Sewell, editors, *POPL'14, San Diego, CA, USA, January 20-21, 2014*, pages 659–670. ACM, 2014. doi:10.1145/2535838.2535886.

5 Beniamino Accattoli and Claudio Sacerdoti Coen. On the relative usefulness of fireballs. In *LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 141–155. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.23.

6 Beniamino Accattoli and Delia Kesner. The structural *lambda*-calculus. In Anuj Dawar and Helmut Veith, editors, *CSL 2010*, volume 6247 of *LNCS*, pages 381–395. Springer, 2010. doi:10.1007/978-3-642-15205-4_30.

7 Beniamino Accattoli and Ugo Dal Lago. Beta reduction is invariant, indeed. In Thomas A. Henzinger and Dale Miller, editors, *CSL-LICS'14, Vienna, Austria, July 14-18, 2014*, pages 8:1–8:10. ACM, 2014. doi:10.1145/2603088.2603105.

8 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.

9 Beniamino Accattoli and Luca Paolini. Call-by-value solvability, revisited. In Tom Schrijvers and Peter Thiemann, editors, *FLOPS 2012, Kobe, Japan, May 23-25, 2012. Proceedings*, volume 7294 of *LNCS*, pages 4–16. Springer, 2012. doi:10.1007/978-3-642-29822-6_4.

10 Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*. Cambridge Tracts in Theoretical Computer Science. CUP, 1999.

11 Thibaut Balabonski. Weak optimality, and the meaning of sharing. In Greg Morrisett and Tarmo Uustalu, editors, *ICFP'13, Boston, MA, USA – September 25-27, 2013*, pages 263–274. ACM, 2013. doi:10.1145/2500365.2500606.

12 Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103. Elsevier, 1984.

13 Zine-El-Abidine Benaissa, Pierre Lescanne, and Kristoffer Høgsbro Rose. Modeling sharing and recursion for weak reduction strategies using explicit substitution. In Herbert Kuchen and S. Doaitse Swierstra, editors, *PLILP'96, Aachen, Germany, September 24-27, 1996, Proceedings*, volume 1140 of *LNCS*, pages 393–407. Springer, 1996. doi:10.1007/3-540-61756-6_99.

14 Gérard Boudol. Computational semantics of term rewriting systems. In M. Nivat and J.C. Reynolds, editors, *Algebraic Methods in Semantics*, page 169–236. CUP, 1985.

15 John R. W. Glauert and Zurab Khasidashvili. Relative normalization in deterministic residual structures. In Hélène Kirchner, editor, *CAAP'96, Linköping, Sweden, April, 22-24, 1996, Proceedings*, volume 1059 of *LNCS*, pages 180–195. Springer, 1996. doi:10.1007/3-540-61064-2_37.

16 Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The geometry of optimal lambda reduction. In Ravi Sethi, editor, *POPL'92, Albuquerque, New Mexico, USA, January 19-22, 1992*, pages 15–26. ACM Press, 1992. doi:10.1145/143165.143172.

17 Delia Kesner. Reasoning about call-by-need by means of types. In *FoSSaCS 2016*, pages 424–441. Springer-Verlag, 2016.

18 Jean-Jacques Lévy. *Réductions correctes et optimales dans le lambda-calcul*. PhD thesis, Université de Paris 7, 1978.

19 Jean-Jacques Lévy. Generalized finite developments. In *Essays in Honour of Gilles Kahn*. CUP, 2007.

20 Luc Maranget. Optimal derivations in weak lambda-calculi and in orthogonal terms rewriting systems. In David S. Wise, editor, *POPL'91, Orlando, Florida, USA, January 21-23, 1991*, pages 255–269. ACM Press, 1991. doi:10.1145/99583.99618.

21 Paul-André Melliès. *Description Abstraite des Systèmes de Réécriture*. PhD thesis, Université Paris 7, december 1996.

22 Paul-André Melliès. Axiomatic rewriting theory II: the $\lambda\sigma$-calculus enjoys finite normalisation cones. *J. Log. Comput.*, 10(3):461–487, 2000. doi:10.1093/logcom/10.3.461.

23 Paul-André Melliès. Axiomatic rewriting theory VI residual theory revisited. In Sophie Tison, editor, *RTA 2002, Copenhagen, Denmark, July 22-24, 2002, Proceedings*, volume 2378 of *LNCS*, pages 24–50. Springer, 2002. doi:10.1007/3-540-45610-4_4.

24 Robin Milner. Local bigraphs and confluence: Two conjectures: (extended abstract). *Electr. Notes Theor. Comput. Sci.*, 175(3):65–73, 2007. doi:10.1016/j.entcs.2006.07.035.

25 Michael J. O'Donnell. *Computing in Systems Described by Equations*, volume 58 of *LNCS*. Springer, 1977. doi:10.1007/3-540-08531-9.

26 H. J. Sander Bruggink. A proof of finite family developments for higher-order rewriting using a prefix property. In Frank Pfenning, editor, *RTA 2006*, volume 4098 of *LNCS*, pages 372–386. Springer, 2006. doi:10.1007/11805618_28.

27 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. CUP, 2003.

28 Vincent van Oostrom. Finite family developments. In Hubert Comon, editor, *RTA-97, Sitges, Spain, June 2-5, 1997, Proceedings*, volume 1232 of *LNCS*, pages 308–322. Springer, 1997. doi:10.1007/3-540-62950-5_80.

29 Vincent van Oostrom, Kees-Jan van de Looij, and Marijn Zwitserlood.] (lambdascope). Workshop on Algebra and Logic on Programming Systems (ALPS), April 2004.

Generalized Refocusing: From Hybrid Strategies to Abstract Machines*

Małgorzata Biernacka[1], Witold Charatonik[2], and Klara Zielińska[3]

1 **Institute of Computer Science, University of Wrocław, Wrocław, Poland**
`mabi@cs.uni.wroc.pl`
2 **Institute of Computer Science, University of Wrocław, Wrocław, Poland**
`wch@cs.uni.wroc.pl`
3 **Institute of Computer Science, University of Wrocław, Wrocław, Poland**
`kzi@cs.uni.wroc.pl`

Abstract

We present a generalization of the refocusing procedure that provides a generic method for deriving an abstract machine from a specification of a reduction semantics satisfying simple initial conditions. The proposed generalization is applicable to a class of reduction semantics encoding hybrid strategies as well as uniform strategies handled by the original refocusing method. The resulting machine is proved to correctly trace (i.e., bisimulate in smaller steps) the input reduction semantics. The procedure and the correctness proofs have been formalized in the Coq proof assistant.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases reduction semantics, abstract machines, formal verification, Coq

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.10

1 Introduction

Refocusing has been introduced by Danvy and Nielsen as a generic procedure to derive an efficient abstract machine from a given reduction semantics [7]. The method has been applied (by hand) to a number of reduction semantics both to derive new machines, and to establish the connection between existing machines and their underlying reduction semantics [3, 10]. Sieczkowski et al. later proposed an axiomatization of reduction semantics sufficient to apply the method and formalized the entire procedure in Coq [17].

However, the refocusing procedure as described previously does not account for a wide class of hybrid rewriting strategies that can be thought of as composed from several different substrategies. Notably, this class includes the normal-order strategy for full normalization in the lambda calculus which is of particular interest due to its use in type checking algorithms for dependently typed programming languages and logic systems, such as Coq [12] and Agda [14].

The problem with applying refocusing to normal order and other similar strategies has been observed by Danvy and Johannsen [6], and by García-Pérez and Nogueira [11], who offer different, partial solutions. The former attribute the problem to backward-overlapping reduction rules in an outermost strategy and propose to apply a correction in the form of "backtracking" in the decomposition procedure. The latter notice that refocusing in this

* This research is supported by the National Science Centre of Poland, under grant number 2014/15/B/ST6/00619.

case becomes context-dependent, and they identify a shape invariant of the context stack they then exploit to transform the machine to an efficient form. Both of these, however, are ad-hoc handmade solutions and do not shed light on how to proceed in a general case.

In this work we propose a generic refocusing procedure that handles both uniform and hybrid rewriting strategies satisfying simple initial conditions. It subsumes Sieczkowski et al.'s formalization [17] that is applicable only to uniform strategies. The procedure is powerful enough to generate realistic abstract machines, particularly machines with environments. It can also be used to verify correctness of existing abstract machines.

Contribution. The specific contributions of our work are as follows:

- We propose a new format for specifying reduction semantics. This format generalizes the standard approach in that it is based on kinded reduction contexts and the ability to restrict composition of reduction contexts based on matching kinds.
- We propose a generalization of refocusing, i.e., an automatic procedure that leads from a specification of a reduction semantics to the corresponding abstract machine. This generalization is enabled by the format of generalized reduction semantics.
- We formalize the generalized refocusing procedure in Coq.
- We prove in Coq the correctness of the abstract machine obtained by refocusing. The correctness of the machine is defined in terms of *tracing* – a variant of bisimulation expressing the property that one system is simulated by another in smaller steps. Specifically, the Coq development contains a proof that the machine obtained by refocusing traces the input reduction semantics.

Related work. On the theoretical side, our work aims at a principled approach to specifying operational semantics, such that facilitates reasoning, analysis, and comparison of various reduction strategies and their composition. The connection made by the formal account of refocusing further enables reasoning about the corresponding abstract machines and possibly their optimizations. This toolbox comes in especially handy when we approach various reduction strategies, reduction semantics, and abstract machines in the existing literature, enabling us to disentangle the different semantic artifacts, expose their underlying structure, and relate one to another. In our work, the issue of hybrid strategies arose directly from the study of existing artifacts, such as abstract machines for full normalization in the lambda calculus that can be proved to implement the normal-order strategy (such as Crégut's abstract machine [4, 11]), or Grégoire & Leroy's strong reduction used to devise an efficient implementation model for Coq [12]; other hybrid strategies are considered in [11, 16].

On the practical side, a natural application of this work is a usable framework for automatic generation of provably correct semantic artifacts, from higher-level semantics to abstract machines. From this point of view our work is related to the existing, much more advanced tools, such as PLT Redex or the K framework, that also facilitate specification of various formats of operational semantics, and experimentation with them [9, 15]. The important difference is that we make explicit the process of producing implementation models for the given high-level specification, while making sure they are provably correct. In the process all the intermediate artifacts can be made available to the user who can further manipulate them. Thus our work can be seen to complement the work on practical tools in the quest to increase reliability of these tools.

Plan of the paper. In Section 2 we recall the basic semantic ingredients needed to describe the original refocusing procedure in its basic form applicable to uniform strategies. In

Sections 3 and 4 we present our main contributions: Section 3 contains our format for specifying reduction semantics, Section 4 – a generalization of the refocusing method to handle hybrid strategies. In Section 5 we discuss the correctness of the generated machine. In Section 6 we briefly describe the implementation, and in Section 7 we conclude.

2 Preliminaries

In this section we recall the fundamental concepts used both in the original setting and in our generalization of the refocusing procedure. We use standard lambda-calculus notions without defining them here (e.g., capture-avoiding substitution, β-reduction).

2.1 Reduction semantics

A *reduction semantics* is a kind of small-step operational semantics, where the positions in a term that can be rewritten are explicitly defined by reduction contexts, rather than implicit in inference rules [8]. More formally, a reduction semantics is a quadruple $\langle \mathcal{T}, \mathcal{C}, \rightharpoonup, V \rangle$, where $\mathcal{T}$ is a set of terms, $\mathcal{C}$ is a set of *reduction contexts*, $\rightharpoonup \subseteq \mathcal{T} \times \mathcal{T}$ is a local rewriting relation (also called *contraction*), and $V \subseteq \mathcal{T}$ is a set of *values*, i.e., terms that represent valid results of computation.

Commonly $\mathcal{C}$ is given by a grammar of contexts. By a *context* we mean a term with exactly one occurrence of a variable called *a hole* and denoted $[\,]$. For a given term t and a context C, by $C[t]$ we denote the result of *plugging* t into C, i.e., the term obtained by substituting the hole in C with t. We then refer to C as a *prefix* of $C[t]$. The $\rightharpoonup$ relation is typically given by a set of rewriting rules. Terms that can be rewritten by $\rightharpoonup$ are called *redices* and those produced by it – *contracta*. We say that a pair $\langle C, t \rangle$ is a *decomposition* of the term $C[t]$ into the context C and the term t.

The *reduction relation* $\to \subseteq \mathcal{T} \times \mathcal{T}$ in the reduction semantics is defined as the compatible closure of contraction: $t_1 \to t_2$ if and only if there exist $C \in \mathcal{C}$, $t_1', t_2' \in \mathcal{T}$ such that $t_1 = C[t_1']$, $t_2 = C[t_2']$, and $t_1' \rightharpoonup t_2'$. *Evaluation* is then defined as the reflexive-transitive closure of the reduction relation (written $\to^*$), and *normal forms* as terms that cannot be reduced. We require the set of values to be a subset of normal forms. Normal forms that are not values are called *stuck terms*.

A *grammar of contexts* consists of a set N of nonterminal symbols, a starting nonterminal, a set S of variables denoting syntactic categories and a set P of productions. All these sets must be finite. In this paper each production has the form $C \to \tau$ where $C \in N$ and τ is a term with free variables in $N \cup \{[\,]\} \cup S$, with exactly one occurrence of a variable from $N \cup \{[\,]\}$. Figure 1 contains an example of a grammar of contexts. We will use a convention that starting nonterminal symbols in grammars are underlined. Intuitively, the restriction on the form of production rules says that, if we interpret the grammar over the alphabet $\{\tau\,[\,] \mid \text{for some } k, k' \text{ there is a production } k \to \tau[k'] \text{ in } G\}$, it generates a regular set of words (as opposed, e.g., to context-free grammars). Then the automaton directly corresponding to the grammar reads the generated context in an outside-in manner, from the topmost symbol towards the hole.

By an *instance of a production* in P we mean the result of replacing in this production occurrences of variables from S with terms from the corresponding syntactic categories (where different occurrences of the same variable may be replaced with different terms). Note that there may be infinitely many instances of productions (because there may be infinitely many terms). The right-hand sides of instances of productions, with nonterminals replaced by holes, are called *elementary contexts*.

$$\begin{array}{rlrl} \text{(terms)} & t ::= x \mid \lambda x.\, t \mid t\, t & \text{(values)} & v ::= \lambda x.\, t \\ \text{(redices)} & r ::= (\lambda x.t)\, t & \text{(reduction contexts)} & \underline{C} ::= [\,] \mid C\, t \end{array}$$

$$(\beta\text{-contraction}) \quad (\lambda x.t_1)\, t_2 \rightharpoonup t_1[t_2/x]$$

Figure 1 The call-by-name reduction semantics for the lambda calculus.

$$(\lambda x.xx)s \xrightarrow{[\,]} ss \xrightarrow{([\,]\,(\lambda x.x))\,s} (\lambda y.y)(\lambda x.x)s \xrightarrow{[\,]\,s} (\lambda x.x)s \xrightarrow{[\,]} s \xrightarrow{[\,]\,(\lambda x.x)} (\lambda y.y)(\lambda x.x) \xrightarrow{[\,]} \lambda x.x$$

where $s = (\lambda x\; y.y)\; a\; (\lambda x.x)$ and a is any closed term

Figure 2 An example evaluation in the call-by-name strategy with reduction contexts indicated above reduction arrows.

Call-by-name strategy. As an example, let us consider the reduction semantics realizing the call-by-name reduction strategy for the lambda calculus. Figure 1 presents the grammar of terms, values and redices in this strategy, and the standard β-contraction relation. A reduction context is either the empty context $[\,]$ or a hole applied to a sequence of terms – as defined by the grammar, where $C\, t$ means that a context generated by C is applied to a term t. The process of evaluation prescribes that at each step we reduce the given term after we decompose it into a redex and a reduction context, unless the term is already a value or a stuck term. It can be seen that, in our setting, terms of the form $x\, t_1 \ldots t_n$ are stuck terms, as they are not proper values nor can they be further reduced. Under call by name, when we evaluate complete programs (which are closed terms), stuck terms cannot occur. Figure 2 shows an example evaluation in the call-by-name reduction semantics.

It is natural to require that a grammar has the *unique-decomposition property* meaning that for each term there exists at most one decomposition into a reduction context and a redex. A grammar with this property implicitly determines a reduction strategy understood as a function that for a given term finds a position of a redex in the term. In the following we will be interested in an explicit description of a lower-level strategy for finding redices. In the example of call-by-name semantics, this strategy prescribes that when we encounter an application, we should continue searching in the left subterm; and when we encounter a lambda abstraction (for which there is no corresponding production in the grammar), we should backtrack. Note that the decision what to do is based only on immediately available and local information (here: the topmost symbol of the processed term; in call by value also on some direct subterms being values). Strategies with this property are called *uniform*. Later we will see that in nonuniform strategies such a decision requires additional knowledge – such strategies can be thought of as composed from different substrategies.

Normal-order strategy. In this work we are interested in applying the refocusing procedure to more complex reduction semantics, such as the normal-order strategy in the lambda calculus. This strategy normalizes a term to its full β-normal form (if it exists) by first evaluating it to its weak-head normal form with the call-by-name strategy, and only then reducing subterms of the resulting weak-head normal form with the same strategy. An example evaluation in this strategy is given in Figure 3. The grammar of reduction contexts

$$(\lambda f\ x.\ f\ ((\lambda f\ x.\ x)\ f\ x))\ (\lambda x.\ g\ x) \xrightarrow{[\,]} \lambda x.\ (\lambda x.\ g\ x)\ ((\lambda f\ x.\ x)\ (\lambda x.\ g\ x)\ x) \xrightarrow{\lambda x.[\,]}$$
$$\lambda x.\ g\ ((\lambda f\ x.\ x)\ (\lambda x.\ g\ x)\ x) \xrightarrow{\lambda x.\ g\ ([\,]\ x)} \lambda x.\ g\ ((\lambda x.\ x)\ x) \xrightarrow{\lambda x.\ g\ [\,]} \lambda x.\ g\ x$$

Figure 3 An example reduction in the normal-order strategy with reduction contexts indicated above the arrows.

$$\begin{array}{rll} \text{(terms)} & t ::= \lambda x.\ t \mid x \mid t\ t, & \underline{E} ::= [\,]_E \mid \lambda x.E \mid F\ t \mid a\ E \\ \text{(neutral terms)} & a ::= x \mid a\ v & F ::= [\,]_F \mid F\ t \mid a\ E \\ \text{(normal terms)} & v ::= a \mid \lambda x.\ v & \end{array}$$

Figure 4 A grammar of reduction contexts for the normal-order strategy.

for this semantics is given in Figure 4; here the set $\mathcal{C}$ of reduction contexts is the language generated from the symbol E, and the contraction relation is β-reduction as in the case of call by name.

The grammar of contexts in Figure 4 again describes a strategy for finding redices. This time, however, this strategy is not uniform: we have two substrategies, one for each nonterminal symbol in the grammar. Each substrategy comes with its own kind of hole as introduced in Section 3.2 – the subscript indicates the kind of context that can be built inside the hole. In other words, if we want to extend a reduction context with a hole of kind k by plugging another context in it, this new context has to be derivable from the nonterminal k. The two presented substrategies agree on reacting to application symbols, but they differ on lambda abstractions. In the case of application symbols they say that when we encounter an application for the first time, we should continue searching in the left subterm with the F substrategy and when we backtrack from the left subterm (when the left subterm is a neutral term in the syntactic category a), we should continue searching in the right subterm with the substrategy E. In the case of lambda abstraction the E substrategy says that we should continue searching in the body of the abstraction while F says that we should backtrack.

In the following, to distinguish between different substrategies, we will use the notion of a *kind* that can be identified with a nonterminal symbol in the underlying grammar.

Uniform and hybrid strategies. The notion of hybrid strategies has been first used by Sestoft [16] informally, and recently studied by García-Pérez and Nogueira [11] who further distinguish between strategies "hybrid in style" and "hybrid in nature." Here we propose a simple definition of a hybrid strategy (that corresponds to being hybrid in style): it is a reduction strategy induced by a reduction semantics whose grammar of contexts has more than one nonterminal symbol (i.e., more than one kind of contexts). This is in contrast to uniform strategies, where the grammars have exactly one nonterminal symbol.

A uniform strategy allows free composition of elementary contexts. In the case of hybrid strategies the introduction of multiple kinds naturally leads to a restriction that when we compose two contexts, the kind of the internal one must be the same as the kind of the hole in the external one. Therefore free composition of elementary contexts may result in an invalid context when the two kinds do not match.

2.2 Abstract machines

Intuitively, abstract machines are just another form of operational semantics, only defined at a lower level of abstraction. Typically, abstract machines are devised in order to specify a semantics that is both precise and reasonably efficient. Ideally, each step of an abstract machine should be done in constant time, which usually can be achieved by rewriting only topmost symbols of machine configurations. Therefore complex operations such as finding a redex in a term or substituting a term for a variable in another term should be divided into smaller steps.

In our setting abstract machines are a kind of abstract rewriting systems of the form $\langle \mathcal{T}, \rightarrow, S, F \rangle$, where $\mathcal{T}$ is a set of configurations (states), $\rightarrow$ is a rewriting relation over $\mathcal{T}$ (the transition relation), and $S, F \subseteq \mathcal{T}$ are initial and final states, respectively. Moreover, we require that machines are deterministic, i.e., that transition relations are partial functions, and that final states are normal forms w.r.t. $\rightarrow$.

In this paper we work with abstract machines with two kinds of configurations. *Eval* configurations are of the form $\langle t, C \rangle_{\mathcal{E}}$ (or of the form $\langle t, C, k \rangle_{\mathcal{E}}$ if we work with hybrid strategies). A long-term goal of the machine in such a configuration is to evaluate the term $C[t]$; a shorter-term goal is to evaluate the term t (according to the substrategy k). The evaluation of t then starts by searching for a redex inside t. When the evaluation of t is finished returning a value v, the machine moves to a *continue* configuration of the form $\langle C, v \rangle_{\mathcal{C}}$ (or $\langle C, k, v \rangle_{\mathcal{C}}$ in the case of hybrid strategies). Again, a long-term goal of the machine in this configuration is to evaluate the term $C[v]$; a shorter-term goal is to find (by searching inside the context C) a decomposition of $C[v]$ into a new context C' and a new term t', and then to move to the configuration $\langle C', t' \rangle_{\mathcal{E}}$.

2.3 Refocusing in uniform strategies

Given a specification of a reduction semantics, a naive implementation of evaluation in this semantics would consist in repeating the following steps until the processed term is a normal form: (i) decompose the given term into a context and a redex, (ii) contract the redex, and (iii) recompose a new term by plugging the contractum in the context.

Refocusing is a mechanical procedure that optimizes this naive implementation by avoiding the reconstruction of intermediate terms in a reduction sequence. Intuitively, a considerable part of the decomposition work in step (i) simply cancels a considerable part of the recomposition work done in step (iii) of the previous iteration, and we would like to avoid this canceled work.

Refocusing (for uniform strategies) is based on the following general observation: *if we plug the contractum in the context (i.e., reconstruct) and then decompose, we obtain the same decomposition as when we continue decomposing directly from where we are.* Let us look at what happens just after contracting a redex in step (ii). There are two cases to be considered: the next redex can or cannot be found inside the contractum. For example, if we have a term $(\lambda x.(\lambda y.\, t_0)\ t_1\ t_2)\ t_3\ t_4\ \ldots\ t_n$, then it decomposes to the redex $(\lambda x.(\lambda y.\, t_0)\ t_1\ t_2)\ t_3$ and the reduction context $[\,]\ t_4\ \ldots\ t_n$ in the call-by-name strategy. But now if we perform the contraction (β-reduction), we do not need to reconstruct the term $((\lambda y.\, t_0)\ t_1\ t_2)[t_3/x]\ t_4 \ldots\ t_n$ and walk through all $t_4, \ldots, t_n$ again to find the next decomposition. It is enough to decompose the result of the contraction to the redex $((\lambda y.\, t_0)\ t_1)[t_3/x]$ and the context $[\,]\ t_2[t_3/x]$, and to plug this context into the previous one (i.e., $[\,]\ t_4\ \ldots\ t_n$).

In the second case (when the contractum does not contain a redex) we also can avoid reconstruction of the whole term, as we can just backtrack from the current reduction context,

reconstruct a part of the whole term and then decompose this part. For example, consider evaluation of the term $(\lambda x\ y.\ t_0)\ v_1\ ((\lambda x.\ t_2)\ v_3\ t_4)\ t_5\ \ldots\ t_n$. According to the left-to-right call-by-value strategy, it decomposes to the redex $(\lambda x\ y.\ t_0)\ v_1$ and the reduction context $[\,]\ ((\lambda x.\ t_2)\ v_3\ t_4)\ t_5\ \ldots\ t_n$. Now after the contraction we can reconstruct a small part of the whole term, namely $(\lambda y.\ t_0)[v_1/x]\ ((\lambda x.\ t_2)\ v_3\ t_4)$, leaving $[\,]\ t_5\ \ldots\ t_n$ as the reduction context and perform a decomposition of this part as before. So as a result we get the redex $(\lambda x.\ t_2)\ v_3$ and the reduction context $(\lambda y.\ t_0)[v_1/x]\ ([\,]\ t_4)\ t_5\ \ldots\ t_n$, and to obtain this decomposition we do not have to traverse all the terms $t_5\ \ldots\ t_n$.

These two ways of avoiding full reconstruction of intermediate terms during evaluation are the main idea behind refocusing. Specifically, a step of a machine generated by refocusing either implements the behavior described above, or it performs the contraction of a found redex. We omit a detailed presentation of the machine since it can be obtained from the one constructed in Section 4.2 by removing all kind information.

Unfortunately, the property emphasized above that underlies the original refocusing procedure no longer holds for hybrid strategies.

2.4 Refocusing in hybrid strategies

Before explaining the problems concerning refocusing in hybrid strategies, we need to discuss some aspects of the internal representation of contexts in implementations of reduction semantics. In general, reduction contexts of a reduction semantics coincide with evaluation contexts that are obtained by defunctionalizing continuations of a CPS-based interpreter implementing the semantics [1, 5]. This way one obtains a common "inside-out" representation of evaluation contexts, where a context is represented by a stack with topmost elements corresponding to the nearest surroundings of the hole and the bottommost – to the root of the context. This representation is also utilized by the refocusing procedure.

In stack representations of contexts the elements of a stack are elementary contexts introduced in Section 2.1. The meaning of such a stack is given by a *recompose* (or *plug*) function that recursively defines the result of inserting a term in the hole of a context. In the case of inside-out contexts this function is a left fold applied to an atomic recomposition function for elementary contexts.

The original refocusing procedure cannot be applied to hybrid strategies, because one of its prerequisites is that the underlying grammar has just one nonterminal symbol. As an example that backs this claim let us consider the strategy defined in Figure 5 and a configuration $\langle c\ t,\ C\rangle_{\mathcal{E}}$. In this configuration, according to E-substrategy the term $c\ t$ cannot be further decomposed and the machine should backtrack moving to the configuration $\langle C,\ c\ t\rangle_{\mathcal{C}}$. On the other hand, according to F-substrategy there might be further redices in t, so the machine should push the elementary context $c\ [\,]$ to the stack and start evaluating t, moving to the configuration $\langle t,\ c\ [\,] :: C\rangle_{\mathcal{E}}$. The choice of the substrategy depends on an occurrence of b in C. If the elementary context $b\ [\,]$ is present somewhere in the stack, the machine should choose the F-substrategy; otherwise, if $b\ [\,]$ is not present in the stack, it should choose the E-substrategy. This, however, requires additional knowledge of what is stored on the stack; a machine generated in the original refocusing procedure has only access to the very local information about the topmost symbol on the stack and the topmost symbol in the processed term (or, in a more general setting, to a bounded initial prefix of the stack and of the term), and simply does not collect this additional knowledge. It is not difficult to see that the strategy induced by this grammar is hybrid in nature – there does not exist a grammar with one nonterminal symbol generating the same set of reduction contexts.

A similar problem occurs when we deal with the normal-order strategy, where a generated machine in a state $\langle \lambda x.\ t,\ C\rangle_{\mathcal{E}}$ has to decide whether it should decompose the lambda

Strategy:

$\underline{E} ::= a\ E \mid b\ F$

$F ::= [\,] \mid a\ F \mid c\ F$

Context $C = \ldots a\ a\ [\,]$:

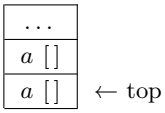


Figure 5 A small reduction strategy and a stack representation of a context.

abstraction. In the particular case of the normal-order strategy the problem can be solved ad hoc: the topmost elementary context in C always indicates the substrategy that should be used, and, in fact, this observation was used e.g. in [6] or [11]. The novelty of our approach is that it works not only in the case of normal-order strategy, but also in a much wider class of hybrid strategies. In Section 3.3 we give an example of a realistic language where the problem discussed above indeed occurs. Moreover, it cannot be solved by the ad hoc method that works for the normal-order strategy, because the choice of a substrategy depends on the occurrence of the ! symbol on the stack.

3 Reduction semantics revisited

In this section we present the first contribution of this work: a generalization of the concepts presented in Section 2.1 that enables refocusing for hybrid strategies.

3.1 Generalized reduction contexts

Our solution to the problem highlighted in Section 2.4 is based on changing the representation of evaluation contexts in generated machines. Specifically, we are going to represent contexts by sequences of instances of productions that generate the represented context. Note that this is not a big change in the representation of contexts from Section 2.4. There a context is represented as a stack (that is, a sequence) of elementary contexts, which are instances of right-hand sides of productions. Since a uniform grammar contains only one nonterminal symbol, this symbol (which is the left-hand side of each production) is implicitly present in each element of the stack. So now the only change in the representation of contexts is that we add an explicit occurrence of the left-hand side of the respective production rule to each element of the stack.

For example, consider the following derivation of the context $\lambda z.\ ((x\ y)\ [\,]_E\ z)$ in the grammar of contexts from Figure 4:

$$E \ \to\ \lambda z.\, E \ \to\ \lambda z.\, (F\ z) \ \to\ \lambda z.\ ((x\ y)\ E\ z) \ \to\ \lambda z.\ ((x\ y)\ [\,]_E\ z)\ .$$

Figure 6 shows the representation of this context. There is one subtlety here: for efficiency reasons (it is always better to pop one rather than two symbols from a stack) the last production of the form $k \to [\,]_k$ is not stored on the stack – it is better to store the nonterminal k in the internal state of the generated abstract machine.

In our generalization of refocusing the additional nonterminal symbol indicates the substrategy that should be used in a given place. Recall the example from Figure 5. A generalized stack representation of the context C with the topmost symbol $\langle E, a\ [\,]\rangle$ indicates that the E-substrategy should be used. It can be seen that this symbol is on the top of stack if and only if there is no occurrence of b in C. Similarly, $\langle F, a\ [\,]\rangle$ indicates F-substrategy;

$E,$	$\lambda z.\,[\,]_E$	
$E,$	$[\,]_F\ \ z$	
$F,$	$(x\ y)\ [\,]_E$	$\leftarrow$ top

Figure 6 The representation of the context $\lambda z.\ \big((x\ y)\ [\,]_E\ z\big)$ w.r.t. the grammar in Fig. 4.

this symbol is on top of stack iff there is an occurrence of b in C. Thus nonterminal symbols stored on stack give the additional knowledge required to choose the appropriate substrategy.

3.2 Grammars of contexts

Here we specify conditions on grammars of contexts required by the generalized refocusing procedure. We first extend the definition of a grammar from Section 2.1 by introducing holes of different kinds (recall that kinds are nonterminal symbols in the grammar) and then restrict the form of production rules.

▶ **Definition 1.** A grammar of contexts G is called *normal* if

1. for each production $k \to \tau$ in the grammar, either τ is a hole or
   - **a.** τ does not contain a hole, and
   - **b.** the unique occurrence of a nonterminal symbol in τ is a direct subterm of τ, and
2. for each nonterminal k, there is a production $k \to [\,]_k$.

Intuitively, a normal grammar generates a prefix-closed set of contexts (i.e., a prefix of a context is also a context). The restriction about direct subterms will be used by the generated abstract machine, which will be analyzing just one symbol at a time. This condition is not crucial and it is not difficult to provide a more general setting in which an abstract machine has access to a bounded initial prefix of the stack and bounded initial prefix of the processed term. In such a setting more strategies can be qualified as uniform. Note also that we introduce here holes indexed with nonterminal symbols – this will be used to keep track of which nonterminal generates given occurrence of a hole. We will refer to $[\,]_k$ as *the hole of kind* k.

We will also require that grammars have the unique-decomposition property. Since this is a semantic restriction that is difficult to check for a given grammar, in Section 4.1 we will pose additional restrictions, in the form of existence of some orders on productions, that entail this property.

3.3 Generalized reduction semantics

In our implementation of the refocusing procedure we use a more flexible definition of reduction semantics than the one given in Section 2.1. It allows to define different kinds of contractions to be used in different kinds of holes in contexts. Thus a generalized reductions semantics is a family of reduction semantics over one language that allows to reduce a term if any sub-semantics allows it.

▶ **Definition 2.** A tuple of the form $\langle \mathcal{T}, K, \{\mathcal{C}_k\}_{k\in K}, \{\rightharpoonup_k\}_{k\in K}, \{V_k\}_{k\in K}\rangle$, where

- $\mathcal{T}$ is a language,
- K is a set of reduction kinds, and
- $\langle \mathcal{T}, \mathcal{C}_k, \rightharpoonup_k, V_k\rangle$ is a reduction semantics for each $k \in K$,

$\mathcal{T} = t ::= x \mid \lambda x.\, t \mid t\ t \mid !t$

$K = \{E,\ F\}$

$(\lambda x.\, t_1)\ t_2 \rightharpoonup_E t_1[t_2/x]$

$!v \rightharpoonup_E v$

$(\lambda x.\, t)\ v \rightharpoonup_F t[v/x]$

$!t \rightharpoonup_F t$

$C \in \mathcal{C}_k \iff C[[\,]_k] \in \mathcal{L}(E) \quad \text{for } k \in K$

$\underline{E} ::= [\,]_E \mid E\ t \mid !F$

$F ::= [\,]_F \mid F\ t \mid v\ F$

$V_E = w ::= a \mid \lambda x.\, t$

$V_F = v ::= b \mid \lambda x.\, t$

$a ::= x \mid a\ t$

$b ::= x \mid b\ v$

Figure 7 Lambda calculus with the call-by-name strategy and a strictness operator.

is called a *generalized reduction semantics*. A term t_1 can then be *reduced* to a term t_2 in this semantics, written $t_1 \to t_2$, if $t_1 = C[s_1]$, $t_2 = C[s_2]$ and $s_1 \rightharpoonup_k s_2$ for some $k \in K$, $s_1,\ s_2 \in \mathcal{T}$ and $C \in \mathcal{C}_k$.

Notions given for reduction semantics generalize in a straightforward way to the generalized ones. We call elements of sets $\mathcal{C}_k$ and V_k, respectively, *(reduction) contexts with holes of kind* k and *values of kind* k. The contraction $\rightharpoonup_k$ is called *k-contraction* and terms that can be rewritten by $\rightharpoonup_k$ are called *redices of kind* k. A term t is called a *potential redex of kind* k if t is not a value of kind k and in every decomposition $t = C[t']$ of t with a nonempty context $C \in \mathcal{C}_k$, the subterm t' is a value of kind k. Intuitively, each non-value term has a (unique) decomposition into a reduction context and a potential redex.

To define families $\{\mathcal{C}_k\}_{k \in K}$ we will use normal grammars of contexts. Formally, we require that the set of context nonterminals coincides with the set K of context kinds and that $C \in \mathcal{C}_k$ if and only if $C[[\,]_k]$ is generated by the grammar.

In Figure 7 we give an example that makes use of the additional expressiveness given by generalized reductions semantics. It presents a lambda calculus with the call-by-name strategy and an extra operator ! that switches the strategy to left-to-right call by value. Here E-substrategy corresponds to call by name and F – to call by value. Note that here E-contraction is different from F-contraction.

4 Generalized refocusing

In this section we present the second main contribution of this work: a generalization of the refocusing method.

4.1 Input to the refocusing procedure

In order to run the generalized refocusing procedure, the user needs to define generalized reduction semantics together with a strategy for searching redices. Given this input, the procedure generates an abstract machine.

We now make precise the requirements on the user input. For a set of productions P, notation $P^{k_1}_{k_2}$ stands for the set of instances of productions from P of the form $k_1 \to ec[k_2]$, where ec is an elementary context. Similarly, P^k stands for instances of the form $k \to \ldots$, that is, with the nonterminal k on the left-hand side of the arrow. We say that an instance $k_1 \to ec[k_2]$ is *compatible* with t if ec is a prefix of t.

▶ **Definition 3.** An *input for generalized refocusing* is composed from

- a generalized reduction semantics $\langle \mathcal{T}, K, \{\mathcal{C}_k\}_{k\in K}, \{\rightharpoonup_k\}_{k\in K}, \{V_k\}_{k\in K}\rangle$ where the family $\{\mathcal{C}_k\}_{k\in K}$ is given by a normal grammar of contexts with the set of productions P,
- for each $k \in K$ and $t \in \mathcal{T}$, a linear strict order $<_{k,t}$ on instances of productions from P^k that are compatible with t, and
- computable functions $\Downarrow$: $\mathcal{T} \times K \to P \times \mathcal{T} \cup \{\mathtt{R}, \mathtt{V}\}$ and $\Uparrow$: $\prod_{k_1,k_2\in K} \left(P^{k_1}_{k_2} \times V_{k_2}\right) \to P \times \mathcal{T} \cup \{\mathtt{R}, \mathtt{V}\}$ (where R tags redices and V tags values)

such that all of the following conditions are satisfied:

1. All $\rightharpoonup_k$ are partial computable functions.
2. Instances of productions $k \to ec_1[k_1]$ and $k \to ec_2[k_2]$ are comparable by $<_{k,t}$ if and only if ec_1, ec_2 are different prefixes of t.
3. If $(k{\to}ec_1[k_1]) <_{k,t} (k{\to}ec_2[k_2])$ and $t = ec_2[v]$, then v is a value of kind k_2.
4. If the order $<_{k,t}$ is empty, then the value of $\Downarrow (t, k)$ is either R or V, depending on whether t is a potential redex or a value of kind k. Otherwise, if the order is nonempty, the value of $\Downarrow (t, k)$ is the greatest element $k \to ec[k']$ in this order, together with the subterm t' of t such that $t = ec[t']$.
5. If the instance $k_1{\to}ec[k_2]$ is the least element in the order $<_{k_1,ec[v]}$, then the value of $\Uparrow (k_1, k_2, (k_1{\to}ec[k_2]), v)$ is either R or V, depending on whether $ec[v]$ is a potential redex or a value of kind k. Otherwise, if $k_1{\to}ec[k_2]$ is not the least element, the value of $\Uparrow (k_1, k_2, (k_1{\to}ec[k_2]), v)$ is the predecessor $k_1{\to}ec'[k'_2]$ of $k_1{\to}ec[k_2]$ in the order $<_{k_1,ec[v]}$, together with the subterm t' of $ec[v]$ such that $ec[v] = ec'[t']$.

Note that the introduced orders describe a depth-first search order of evaluation. More precisely, suppose that $(k{\to}ec_1[k_1]) <_{k,t} \ldots <_{k,t} (k{\to}ec_n[k_n])$ are all productions in P^k compatible with t, sorted in the order $<_{k,t}$. Then for some terms $t_1, \ldots, t_n$ we have $t = ec_1[t_1] = \ldots = ec_n[t_n]$. Condition (4) says that k-substrategy starts searching for redices in term t_n. Condition (5) says that in consecutive iterations (for $i = n-1, \ldots, 1$) this strategy moves the search from t_{i+1} to t_i, and condition (3) says that it happens only after completely traversing t_{i+1} and finding it to be a value. The functions $\Downarrow$, $\Uparrow$ describe a DFS order of evaluation in a computable way. They also determine if a term is a potential redex or (after checking all possible decompositions w.r.t. a given substrategy) if it is a value of a given kind.

It is worth noting that if we want the generated abstract machine to execute each step in constant time, all of the mentioned computable functions should also work in constant time.

We give an example input for the generalized refocusing procedure in Figures 7 and 8. The former contains a generalized semantics and the latter contains an order and functions $\Downarrow$ and $\Uparrow$. The functions are presented in a relational style, with arguments on the left-hand side and results on the right-hand side of the arrows. The notation $[t_1]_{k_1}$ stands for (t_1, k_1), the notation $\left[ec\ [v]_{k_2}\right]_{k_1}$ for $(k_1, k_2, (k_1{\to}ec[k_2]), v)$ and the notation $ec\ [t_2]_{k_3}$ on a right-hand side of the arrow $\Uparrow$ involving k_1 as the leftmost argument stands for $((k_1{\to}ec[k_3]), t_2)$.

The result of $[t_1]_{k_1} \Downarrow$ prescribes how to proceed when a term t is considered by a k_1-substrategy for the first time. If it returns $ec\ [t_2]_{k_2}$, the search should be continued in the subterm t_2 by the k_2-substrategy and the current evaluation context should be extended by ec. Otherwise there is no further decomposition of t and it is determined if it is a value or a potential redex. For example, $[!t]_E \Downarrow !\ [t]_F$ prescribes that when the E-substrategy encounters a term $!t$, it has to decompose t according to the F-substrategy with the current evaluation contexts extended by $!\,[\,]$.

The result of $\left[ec_1\ [v]_{k_2}\right]_{k_1} \Uparrow$ defines how to proceed when a term $ec_1[v]$ is reconsidered by a k_1-substrategy after completing the search in the subterm v w.r.t. the k_2-substrategy and finding that it is a value of kind k_2. The options are the same as in the case of $\Downarrow$.

The order on instances of productions:
$(F \to v\ F) <_{F,(v\ t)} (F \to F\ t)$ for all values v of kind F and all terms t

Functions $\Downarrow$ and $\Uparrow$, where $k \in \{E, F\}$:

$$
\begin{array}{rr}
[x]_k \Downarrow \mathtt{V} & [\ [\lambda x.t_1]_E\ t_2]_E \Uparrow \mathtt{R} \\
[\lambda x.t]_k \Downarrow \mathtt{V} & [\ [a]_E\ t]_E \Uparrow \mathtt{V} \\
[t_1\ t_2]_k \Downarrow\ [t_1]_k\ t_2 & [!\ [v]_F\ ]_E \Uparrow \mathtt{R} \\
[!t]_E \Downarrow !\ [t]_F & [\ [v]_F\ t]_F \Uparrow v\ [t]_F \\
[!t]_F \Downarrow \mathtt{R} & [\lambda x.t\ [v]_F\ ]_F \Uparrow \mathtt{R} \\
 & [b\ [v]_F\ ]_F \Uparrow \mathtt{V}
\end{array}
$$

Figure 8 An input for generalized refocusing that formalizes the semantics from Figure 7.

4.2 Output from the refocusing procedure

The machines obtained by the generalized refocusing procedure have the form given in the definition below. The notion $ConRep_{G,k}$ in this definition is the set of generalized stack representations (cf. Section 3.1) of contexts in $\mathcal{C}_k$ w.r.t. a grammar G. To distinguish contexts and their representations we use the typewriter font for the latter. The configuration labels $\mathcal{E}$, $\mathcal{C}$ historically stand for "eval" and "continue".

▶ **Definition 4.** For a given input as in Definition 3, the *machine generated by generalized refocusing* is $\langle \mathcal{S}, \to, \{\langle t, \varepsilon, k_0\rangle_{\mathcal{E}} \in \mathcal{S}\},\ \{\langle \varepsilon, k_0, v\rangle_{\mathcal{C}} \in \mathcal{S}\}\rangle$, where

$$\mathcal{S} = \{\ \langle t, \mathtt{C}, k\rangle_{\mathcal{E}}, \langle \mathtt{C}, k, v\rangle_{\mathcal{C}} \mid t \in \mathcal{T}, k \in K, \mathtt{C} \in ConRep_{G,k}, v \in V_k\}$$

and $\to$ is defined as follows.

$$
\begin{array}{ll}
\langle t, \mathtt{C}, k\rangle_{\mathcal{E}} \to \langle \mathtt{C}, k, t\rangle_{\mathcal{C}} & \text{if } [t]_k \Downarrow \mathtt{V}, \\
\langle t_1, \mathtt{C}, k\rangle_{\mathcal{E}} \to \langle t_2, \mathtt{C}, k\rangle_{\mathcal{E}} & \text{if } [t]_k \Downarrow \mathtt{R} \wedge t_1 \rightharpoonup_k t_2, \\
\langle t_1, \mathtt{C}, k_1\rangle_{\mathcal{E}} \to \langle t_2, (k_1, ec) :: \mathtt{C}, k_2\rangle_{\mathcal{E}} & \text{if } [t_1]_{k_1} \Downarrow ec\ [t_2]_{k_2}, \\
\langle (k_1, ec) :: \mathtt{C}, k_2, v\rangle_{\mathcal{C}} \to \langle \mathtt{C}, k_1, ec[v]\rangle_{\mathcal{C}} & \text{if } \left[ec\ [v]_{k_2}\ \right]_{k_1} \Uparrow \mathtt{V}, \\
\langle (k_1, ec) :: \mathtt{C}, k_2, v\rangle_{\mathcal{C}} \to \langle t, \mathtt{C}, k_1\rangle_{\mathcal{E}} & \text{if } \left[ec\ [v]_{k_2}\ \right]_{k_1} \Uparrow \mathtt{R} \wedge ec[v] \rightharpoonup_{k_1} t, \\
\langle (k_1, ec_1) :: \mathtt{C}, k_2, v\rangle_{\mathcal{C}} \to \langle t, (k_1, ec_2) :: \mathtt{C}, k_3\rangle_{\mathcal{E}} & \text{if } \left[ec_1\ [v]_{k_2}\ \right]_{k_1} \Uparrow ec_2\ [t]_{k_3}.
\end{array}
$$

In fact, in configurations of the form $\langle \mathtt{C}, k, v\rangle_{\mathcal{C}}$ symbol k has no computational meaning and could be safely removed. We leave it here for the sake of simplicity.

Note that the construction of the machine does not depend on conditions (1)–(5) of Definition 3, it depends only on the functions $\Downarrow$, $\Uparrow$ and contractions. The conditions are needed to guarantee that the constructed machine realizes the input reduction semantics.

5 Correctness of the generated machine

In [17] the authors prove that the original refocusing procedure generates an abstract machine that is extensionally equivalent to the input semantics. Formally, a term t can be reduced (possibly in many steps) to a value v w.r.t. an input reduction semantics if and only if the

generated machine started in $\langle t, \varepsilon\rangle_{\mathcal{E}}$ evaluates to $\langle \varepsilon, v\rangle_{\mathcal{C}}$. Here we show a stronger property, namely that the generated machine exactly implements the reduction semantics given as input, possibly in smaller steps. This idea is captured in the following definition of *tracing*.

▶ **Definition 5.** An abstract rewriting system $\langle \mathcal{T}, \to\rangle$ **traces** another system $\langle \mathcal{S}, \Rightarrow\rangle$ if there exists a surjection $[\![]\!] : \mathcal{T} \to \mathcal{S}$ such that

1. if $t_1 \to t_2$, then $[\![t_1]\!] = [\![t_2]\!]$ or $[\![t_1]\!] \Rightarrow [\![t_2]\!]$,
2. if $s_1 \Rightarrow s_2$, then for each t_0 such that $[\![t_0]\!] = s_1$ there exists a sequence $t_0 \to \ldots \to t_{n+1}$, where $[\![t_0]\!] = \ldots = [\![t_n]\!]$ and $[\![t_{n+1}]\!] = s_2$, and
3. there are no infinite sequences $t_0 \to t_1 \to \ldots$, where $[\![t_n]\!] \not\Rightarrow [\![t_{n+1}]\!]$ for all n (i.e., there are no silent loops).

The definition is similar to the one used by Hardin et al. to extract reduction strategies in a calculus of closures from virtual machines [13]. Although in this paper we work only with deterministic systems, it is worth mentioning that our definition, as opposed to the one from [13], is also adequate for nondeterministic systems.

The following theorem states that each generated machine traces the input reduction semantics. It is proved in Coq, the proof can be found in the implementation in file `refocusing/refocusing_machine_facts.v`.

▶ **Theorem 6.** *Let M be the machine generated by generalized refocusing procedure from a reduction semantics with the set of terms $\mathcal{T}$ and the reduction relation $\to$. Then M traces $\langle \mathcal{T}, \to\rangle$.*

6 Implementation

Our implementation of generalized refocusing is available at the repository `http://bitbucket.org/kzi/generalized_refocusing`. The current (at the moment of writing this article) version of the code, including examples, consists of over 8000 lines of definitions and proofs. It was tested in Coq versions 8.5 and 8.6. The repository contains several examples, including a novel derivation of a machine with an environment for full β-normalization from a language with explicit substitutions (see file `lam_cl_es_no.v`). We suggest to start with simpler examples of lambda calculus with the call-by-name and call-by-value strategies in files `cbn_lam.v` and `cbv_lam.v`. A formalization of lambda calculus with the call-by-name strategy and the strictness operator from Figure 7 can be found in `cbn_strict.v` file. In three cases: MiniML, lambda calculus with the normal-order strategy, and lambda calculus with the normal-order strategy and simple explicit substitutions the examples also contain a manually defined machine and a proof of equivalence between this machine and the automatically generated one.

In the following we briefly describe the implementation of the semantics provided by a user as input to generalized refocusing, as defined in Definition 3. The user needs to define two modules: one that implements the reduction semantics (which satisfies the signature `PRE_REF_SEM`), and one that implements the lower-level reduction strategy (which satisfies the signature returned by `REF_STRATEGY`). Based on these modules, the functor `RedRefSem` produces a module defining the refocusable semantics; in particular, it automatically proves crucial decomposition lemmas required by refocusing. Then the functor `RefEvalApplyMachine` applied to the refocusable semantics module produces an abstract machine. Finally, the functor `RefEvalApplyMachine_Facts` applied to the semantics and the machine provides the proof of Theorem 6. The proof itself is an instance of `RW_TRACING` type-class defined in the file `rewriting_system/rewriting_system_tracing.v`.

We show an excerpt of the relevant signatures in Figure 9. In the semantics module, the parameters (such as terms, kinds, potential redices, or values) are provided by the user, and their types reflect their dependence on kinds. There is some technical burden caused by Coq here: the user has to define values and potential redices as types disjoint from terms, and provide injections from these types into terms. We also require the user to define elementary contexts that depend on two kinds: the first one is the nonterminal on the left-hand side of the underlying production, and the second one is the kind of the hole (this is the nonterminal occurring in the right-hand side of the production). The reduction contexts can then be represented as lists of elementary contexts that agree on kinds, as described in Section 3.1. In addition, the user has to provide proofs for the axioms ensuring that potential redices and values can have only trivial decompositions, and that values cannot be potential redices.

The implementation of the strategy consists in defining two functions: `dec_term` and `dec_context` that correspond to the $\Downarrow$ and $\Uparrow$ functions, respectively. The types of these functions make use of the `elem_dec` type of elementary decompositions. The user then needs to specify the search order on elementary contexts together with proofs that this order is well founded, transitive and that only and all immediate prefixes of a term are comparable. With these properties in place the user proceeds to proving properties required by Definition 3. The axioms at the bottom of the listing in Figure 9 correspond to these properties (due to lack of space some axioms are omitted): `elem_context_det` implements condition 3, `dec_term_term_top` implements the second part of condition 4; `dec_context_red_bot`, `dec_context_val_bot` and `dec_context_term_next` correspond to condition 5.

7 Conclusion and perspectives

We have proposed a new format for specifying reduction semantics that utilizes grammars of kinded reduction contexts. Based on this format, we have developed a generic refocusing procedure that handles a wide class of hybrid rewriting strategies satisfying simple initial conditions. The procedure is formalized in Coq proof assistant and proved to correctly translate reduction semantics to abstract machines. The original refocusing procedure, as described in [17], can be obtained from the proposed here by restricting the set of context kinds to a singleton and optimizing out dependencies on its elements.

The current formalization has several limitations that we plan to address in future work. For example, it does not account for context-sensitive reduction strategies – such as those defining control operators – where the contraction function takes into account the current reduction context (this is different from the situation described in this paper where the *decomposition* function depends on the kind of the current reduction context). Moreover, it is not possible to define layered reduction semantics that would lead to an abstract machine with several layers of stacks representing contexts (such as 2-layered semantics for the control operators shift and reset[2]). A limitation of a different nature is that refocusing is based on the assumption that the semantics is deterministic. It would be interesting to see how this restriction could be relaxed and lead to a systematic way of deriving nondeterministic abstract machines.

Acknowledgments. We would like to thank the anonymous reviewers of FSCD for their helpful and insightful comments.

```
Module Type PRE_REF_SEM.

  Parameters (term ckind : Set) (init_ckind : ckind) (redex value : ckind → Set).
  Parameter contract : ∀ {k}, redex k → option term.
  Parameters (elem_context_kinded : ckind → ckind → Set)
             (elem_plug : ∀ {k0 k1}, term → elem_context_kinded k0 k1 → term).
  Notation "ec :[ t ]" := (elem_plug t ec) (at level 0).

  Inductive context (k1 : ckind) : ckind → Set :=
  | empty : context k1 k1
  | ccons : ∀ {k2 k3} (ec : elem_context_kinded k2 k3), context k1 k2 → context k1 k3.
  Notation "c [ t ]" := (plug t c) (at level 0).

  Definition reduce k t1 t2 := ∃ {k'} (c : context k k') (r : redex k') t,
                               dec t1 k (d_red r c) ∧ contract r = Some t ∧ t2 = c[t].
  Axioms
    (redex_trivial1 : ∀ {k k'} (r : redex k) ec t, ec:[t] = r → ∃ (v : value k'), t = v)
    (value_trivial1 : ∀ {k1 k2} ec {t} (v : value k1), ec:[t]=v → ∃ (v' : value k2), t = v')
    (value_redex : ∀ {k} (v : value k) (r : redex k), value_to_term v <> redex_to_term r).
  ...
End PRE_REF_SEM.

Module Type REF_STRATEGY (S: PRE_REF_SEM).
  Import S.
  Inductive elem_dec k : Set :=
  | ed_red : redex k → elem_dec k
  | ed_dec : ∀ k', term → elem_context_kinded k k' → elem_dec k
  | ed_val : value k → elem_dec k.

  Parameter dec_term: term → ∀ k, elem_dec k.
  Parameter dec_context {k k'} (ec:elem_context_kinded k k') (v:value k') : elem_dec k.
  Parameter search_order: ∀ k, term→ elem_context_in k→ elem_context_in k → Prop.
  Notation "t |∼ ec1 « ec" := (search_order _ t ec1 ec2).

  Definition so_maximal {k} (ec : elem_context_in k) t := ∀ ec', ∼t |∼ ec « ec'.
  Definition so_minimal {k} (ec : elem_context_in k) t := ∀ ec', ∼t |∼ ec' « ec.
  Definition so_predecessor {k} ec0 ec1 t :=
    t |∼ ec0 « ec1 ∧ ∀ (ec : elem_context_in k), t |∼ ec « ec1 → ∼t |∼ ec0 « ec.

  Axioms
    (elem_context_det : ∀ {k0 k1 k2} t ec0 ec1,
                  t |∼ ec0 « ec1 → ∃ (v : value k2), t = ec1:[v])
    (dec_term_term_top : ∀ {k k'} t t' ec,
                  dec_term t k = ed_dec _ t' ec → so_maximal ec t)
    (dec_context_red_bot : ∀ {k k'} (v : value k') {r : redex k} ec,
                  dec_context ec v = ed_red r → so_minimal ec ec:[v])
    (dec_context_val_bot : ∀ {k k'} (v : value k') {v' : value k} ec,
                  dec_context ec v = ed_val v' → so_minimal ec ec:[v])
    (dec_context_term_next : ∀ {k0 k1 k2} (v : value k1) t ec0 ec1,
                  dec_context ec0 v = ed_dec _ t ec1 → so_predecessor ec1 ec0 ec0:[v])
  ...
End REF_STRATEGY.
```

Figure 9 Signatures for refocusable semantics (fragments).

References

1. Mads Sig Ager, Dariusz Biernacki, Olivier Danvy, and Jan Midtgaard. A functional correspondence between evaluators and abstract machines. In Dale Miller, editor, *Proceedings of the Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'03)*, pages 8–19, Uppsala, Sweden, August 2003. ACM Press.
2. Małgorzata Biernacka, Dariusz Biernacki, and Olivier Danvy. An operational foundation for delimited continuations in the CPS hierarchy. *Logical Methods in Computer Science*, 1(2:5):1–39, November 2005.
3. Małgorzata Biernacka and Olivier Danvy. A syntactic correspondence between context-sensitive calculi and abstract machines. *Theoretical Computer Science*, 375(1-3):76–108, 2007.
4. Pierre Crégut. Strongly reducing variants of the Krivine abstract machine. *Higher-Order and Symbolic Computation*, 20(3):209–230, 2007. A preliminary version was presented at the 1990 ACM Conference on Lisp and Functional Programming.
5. Olivier Danvy. Defunctionalized interpreters for programming languages. In Peter Thiemann, editor, *Proceedings of the 2008 ACM SIGPLAN International Conference on Functional Programming (ICFP'08)*, SIGPLAN Notices, Vol. 43, No. 9, Victoria, British Columbia, September 2008. ACM, ACM Press. Invited talk.
6. Olivier Danvy and Jacob Johannsen. From outermost reduction semantics to abstract machine. In Gopal Gupta and Ricardo Peña, editors, *23rd International Symposium on Logic-Based Program Synthesis and Transformation, LOPSTR 2013*, pages 91–98. Springer-VS, 2014. doi:10.1007/978-3-319-14125-1.
7. 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.
8. Matthias Felleisen. *The Calculi of λ-v-CS Conversion: A Syntactic Theory of Control and State in Imperative Higher-Order Programming Languages*. PhD thesis, Computer Science Department, Indiana University, Bloomington, Indiana, August 1987.
9. Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics Engineering with PLT Redex*. The MIT Press, 1st edition, 2009.
10. Ronald Garcia, Andrew Lumsdaine, and Amr Sabry. Lazy evaluation and delimited control. In Benjamin C. Pierce, editor, *Proceedings of the Thirty-Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 153–164. ACM Press, January 2009.
11. A García-Pérez and Pablo Nogueira. On the syntactic and functional correspondence between hybrid (or layered) normalisers and abstract machines. *Science of Computer Programming*, 95:176–199, 2014.
12. Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In Simon Peyton Jones, editor, *Proceedings of the 2002 ACM SIGPLAN International Conference on Functional Programming (ICFP'02)*, SIGPLAN Notices, Vol. 37, No. 9, pages 235–246, Pittsburgh, Pennsylvania, September 2002. ACM, ACM Press.
13. Thérèse Hardin, Luc Maranget, and Bruno Pagano. Functional runtime systems within the lambda-sigma calculus. *Journal of Functional Programming*, 8(2):131–172, 1998.
14. Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, September 2007.
15. Grigore Roşu and Traian Florin Şerbănuţă. An overview of the K semantic framework. *Journal of Logic and Algebraic Programming*, 79(6):397–434, 2010. doi:10.1016/j.jlap.2010.03.012.

16 Peter Sestoft. Demonstrating lambda calculus reduction. In Torben Æ. Mogensen, David A. Schmidt, and I. Hal Sudborough, editors, *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, number 2566 in Lecture Notes in Computer Science, pages 420–435. Springer-Verlag, 2002.

17 Filip Sieczkowski, Małgorzata Biernacka, and Dariusz Biernacki. Automating derivations of abstract machines from reduction semantics: a generic formalization of refocusing in Coq. In Juriaan Hage and Marco T. Morazán, editors, *The 22nd International Conference on Implementation and Application of Functional Languages (IFL 2010)*, number 6647 in Lecture Notes in Computer Science, pages 72–88, Alphen aan den Rijn, The Netherlands, September 2010. Springer-Verlag.

Nested Multisets, Hereditary Multisets, and Syntactic Ordinals in Isabelle/HOL

Jasmin Christian Blanchette[*1], **Mathias Fleury**[2], **and Dmitriy Traytel**[3]

1 **Vrije Universiteit Amsterdam, The Netherlands; and Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany**
2 **Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany**
3 **Institute of Information Security, Department of Computer Science, ETH Zürich, Zürich, Switzerland**

Abstract

We present a collection of formalized results about finite nested multisets, developed using the Isabelle/HOL proof assistant. The nested multiset order is a generalization of the multiset order that can be used to prove termination of processes. Hereditary multisets, a variant of nested multisets, offer a convenient representation of ordinals below ϵ_0. In Isabelle/HOL, both nested and hereditary multisets can be comfortably defined as inductive datatypes. Our formal library also provides, somewhat nonstandardly, multisets with negative multiplicities and syntactic ordinals with negative coefficients. We present applications of the library to formalizations of Goodstein's theorem and the decidability of unary PCF (programming computable functions).

1998 ACM Subject Classification F.3.1 [Logics and Meanings of Programs] Specifying and Verifying and Reasoning about Programs, F.4.1 [Mathematical Logic and Formal Languages] Mathematical Logic I.2.3 [Artificial Intelligence] Deduction and Theorem Proving

Keywords and phrases Multisets, ordinals, proof assistants

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.11

1 Introduction

In their seminal article on proving termination using multisets [15], Dershowitz and Manna introduced two orders of increasing strength. The *multiset order* lifts a base partial order on a set A to finite multisets over A. It forms the basis of the multiset path order, which has many applications in term rewriting [41] and automatic theorem proving [1]. The *nested multiset order* is a generalization of the multiset order that operates on multisets that can be nested in arbitrary ways. Nesting can increase the order's strength: If $(A, <)$ has ordinal type $\alpha < \epsilon_0$, the associated multiset order has ordinal type ω^α, whereas the nested order has ordinal type $\epsilon_0 = \omega^{\omega^{\omega^{\cdot^{\cdot^{\cdot}}}}}$.

In this paper, we present formal proofs of the main properties of the nested multiset order that are useful in applications: preservation of well-foundedness and preservation of totality (linearity). The proofs are developed in the Isabelle/HOL proof assistant [27]. To

* Blanchette has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka).

our knowledge, this is the first development of its kind in any proof assistant. Our starting point is the following inductive datatype of nested finite multisets over a type $'a$ (Section 4):

datatype $'a$ *nmultiset* = Elem $'a$ | MSet (($'a$ *nmultiset*) *multiset*) .

The above Isabelle/HOL command introduces a (unary postfix) type constructor, *nmultiset*, equipped with two constructors, Elem : $'a \to 'a$ *nmultiset* and MSet : ($'a$ *nmultiset*) *multiset* $\to$ $'a$ *nmultiset*, where $'a$ is a type variable and *multiset* is the type constructor of (finite) multisets. Throughout the paper, we will write "multiset" meaning "finite multiset," following Isabelle conventions. The **datatype** command also introduces a recursor that can be used to define primitive recursive functions. In addition, the command provides an induction principle, which allows us to assume, in the MSet *NM* case, that the desired property holds for all nested multisets belonging to *NM*.

The definition of *nmultiset* exhibits recursion through a non-datatype (*multiset*). In recent versions of Isabelle/HOL, recursion is allowed under arbitrary type constructors that are *bounded natural functors* [37, 8], a semantic criterion that is met by bounded sets, multisets, and other functors. This flexibility is absent in other proof assistants. The HOL systems and Lean support quotient types, which can be used to build multisets from lists; in Agda and Coq, setoids can be employed instead [3]. However, we argue that datatypes lead to simpler definitions and proofs than any of the alternatives.

If we omit the Elem constructor, we obtain the hereditary multisets (Section 5):

datatype *hmultiset* = HMSet (*hmultiset multiset*) .

This type is similar to hereditarily finite sets, a model of set theory without the axiom of infinity, but with multisets instead of finite sets. It is easy to embed *hmultiset* in $'a$ *nmultiset*, and using Isabelle's Lifting and Transfer tools [20], we can lift definitions and results from the larger type to the smaller type, such as the definition of the nested multiset order.

Hereditary multisets offer a convenient representation for ordinals below ϵ_0 (Section 6). These are the ordinals that can be expressed syntactically in Cantor normal form:

$$\alpha \ ::= \ \omega^{\alpha_1} \cdot c_1 + \cdots + \omega^{\alpha_n} \cdot c_n \qquad \text{where } c_i \in \mathbb{N}^{>0} \text{ and } \alpha_1 > \cdots > \alpha_n \,.$$

The correspondence with hereditary multisets is straightforward:

$$\alpha \ ::= \ \{\underbrace{\alpha_1, \ldots, \alpha_1}_{c_1 \text{ occurrences}}, \ldots, \underbrace{\alpha_n, \ldots, \alpha_n}_{c_n \text{ occurrences}}\} \,.$$

The coefficients c_i are represented by multiset multiplicities, and the ω exponents are the multiset's members. Thus:

$$\{\} = 0 \qquad \{0\} = \{\{\}\} = \omega^0 = 1 \qquad \{0,0,0\} = \{\{\},\{\},\{\}\} = \omega^0 \cdot 3 = 3$$
$$\{1\} = \{\{\{\}\}\} = \omega^{\omega^0} = \omega^1 = \omega \qquad \{\omega\} = \{\{\{\{\}\}\}\} = \omega^\omega$$

The standard addition and multiplication operations on ordinals are not commutative – e.g., $1 + \omega = \omega \neq \omega + 1$. Instead, we formalized the Hessenberg (or natural) operations [34]. These are more convenient in many applications, and because they share many properties with natural numbers, they are easier to automate in Isabelle.

When carrying out proofs, we sometimes find ourselves wishing that it would be possible to subtract an ordinal from another. To support this, we define a type of signed multisets, or hybrid multisets [2], and use it to represent signed ordinals (Section 7).

We employ our library to formalize two examples that require ordinals or the nested multiset order: Goodstein's theorem (Section 8) and the decidability of unary PCF (Section 9). Together with colleagues, Blanchette also used the library to formalize a variant of the transfinite Knuth–Bendix order [4, 5]. We gave some thought to proof automation, generalizing existing simplification procedures and exploiting Isabelle's arithmetic type classes. Our work also demonstrates the usefulness of bounded natural functors. The Isabelle theory files are available as part of the *Archive of Formal Proofs* [6].

2 Isabelle/HOL

Isabelle is a generic proof assistant whose metalogic is an intuitionistic fragment of polymorphic higher-order logic (simple type theory). Types are built from type variables $'a$, $'b$, ... and type constructors, written infix or postfix (e.g., $\to$, *multiset*). All types are inhabited. Terms t, u are built from variables x, constants c, abstractions $\lambda x.\ t$, and applications $t\ u$. Constants may be higher-order, i.e., they may be of function type. A formula is a term of type *prop*. The metalogical operators are $\bigwedge$, $\Longrightarrow$, and $\equiv$, for universal quantification, implication, and equality. The notation $\bigwedge x.\ t$ is syntactic sugar for $\bigwedge(\lambda x.\ t)$.

Isabelle/HOL is the instantiation of Isabelle with classical higher-order logic (HOL) extended with type classes as its object logic, complete with a Boolean type *bool*, an equality predicate $=$, the usual connectives ($\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\longleftrightarrow$) and quantifiers ($\forall$, $\exists$), and Hilbert's choice. HOL formulas, of type *bool*, are embedded in the metalogic. The distinction between *prop* and *bool* is important operationally, but it is not essential to understand this paper.

Isabelle/HOL offers two primitive definitional mechanisms: The **typedef** command introduces a type that is isomorphic to a nonempty subset of an existing type, and the **definition** command introduces a constant as equal to an existing term. On top of these, Isabelle/HOL provides a rich specification language that includes inductive datatypes and predicates, recursive functions, and their coinductive counterparts, as well as quotient types.

Proofs are expressed either as a sequence of low-level proof methods, called *tactics*, which manipulate the proof state directly, or in a declarative format called Isar [39], which allows tactics only as terminal procedures. We generally prefer the more readable Isar style. The main proof methods are the *simplifier*, which rewrites terms using conditional oriented equations; the *classical reasoner*, which applies introduction and elimination rules; decision procedures for linear arithmetic; and *metis*, a complete first-order prover based on superposition. In addition, the Sledgehammer tool [29] integrates third-party automatic theorem provers. It can be applied to any proof goal. In case of success, it provides a short Isar proof, often using *metis*.

3 Multisets

Multisets over $'a$ are defined in Isabelle's standard library as isomorphic to the set of multiplicity functions f that are 0 at all but finitely many points x:

$$\textbf{typedef}\ 'a\ \mathit{multiset} = \left\{f : 'a \to \mathit{nat} \mid \mathsf{finite}\ \{x \mid f\ x > 0\}\right\}.$$

Values are constructed from the empty multiset $\{\}$ and the $\mathsf{add}\ x\ A$ operation. For concrete multisets, we use standard set notation. The singleton multiset $\{x\}$ is easy to define in terms of add and $\{\}$. Multiset union $\uplus$, which adds the multiplicities of its arguments, is an instance of the polymorphic $+ : 'a \to 'a \to 'a$ operator. The relevant $+$ type classes provide a wealth of lemmas and some proof automation. Other operations include $-$, $\cup$, $\cap$, $\subset$, $\subseteq$, $<$, and $\leq$.

Given a type $'a$ equipped with a partial order $<$, the $<$ operator on multisets corresponds to the Dershowitz–Manna extension [15]. The extension operator is also available as a function $\mathsf{mult} : ('a \times 'a)\ \mathit{set} \to ('a\ \mathit{multiset} \times 'a\ \mathit{multiset})\ \mathit{set}$. It is defined as the transitive closure of the one-step Dershowitz–Manna extension defined by

$$\mathsf{mult1}\ R = \{(A, B).\ \exists y\ B_0\ X.\ B = B_0 + \{y\} \wedge A = B_0 + X \wedge \forall x \in X.\ (x, y) \in R\}\,.$$

For both R and $\mathsf{mult1}\ R$, the smaller value is on the left.

The operation $\mathsf{set} : 'a\ \mathit{multiset} \to 'a\ \mathit{set}$ returns the set of elements in a multiset. The operation $\mathsf{image} : ('a \to 'b) \to 'a\ \mathit{multiset} \to 'b\ \mathit{multiset}$ applies a function elementwise to a multiset. It is defined using an iterator on finite sets, adding each mapped element $f\ x$ with the multiplicity of x in the initial multiset A (given by $\mathsf{count}\ A\ x$):

definition $\mathsf{image} : ('a \to 'b) \to 'a\ \mathit{multiset} \to 'b\ \mathit{multiset}$ **where**
$\mathsf{image}\ f\ A = \mathsf{Finite\_Set.fold}\ (\lambda x.\ (\mathsf{add}\ (f\ x))^{\mathsf{count}\ A\ x})\ \{\}\ (\mathsf{set}\ A)$

The structure $(\mathit{multiset}, \mathsf{image}, \mathsf{set}, \aleph_0)$ forms a bounded natural functor [37, 8]: image is the functorial action, set is the natural transformation, and $\aleph_0$ is an upper bound on the cardinality of the sets returned by set. Induction and recursion *through* a multiset are expressed in terms of set and image. The cardinality bound is needed to construct the datatype as the least fixed point of an isomorphism equation.

Isabelle's datatypes do not allow genuinely negative occurrences in recursion, since this cannot be done consistently in HOL. However, some type constructors support both negative and positive views: $'a\ \mathit{multiset}$ can be viewed negatively as a fragment of $'a \to \mathit{nat}$ or positively as a quotient of $'a\ \mathit{list}$ (with respect to the relation "contains the same elements with the same multiplicities as"). It is this latter view that is fruitful for recursion. It is easy to see that image then corresponds to the functorial action map on lists.

Isabelle's multiset theory is not as developed as other areas, such as lists and sets. Our first contribution has been to introduce some missing concepts and lemmas, which we added either directly to the Isabelle distribution or collected in a theory file in the *Archive of Formal Proofs* [6]. These include a $\mathsf{replicate}\ n\ x$ operator, which constructs the multiset consisting of n copies of x, and the cartesian product $\times$. We also showed that the Huet–Oppen extension [19] of a partial order coincides with the Dershowitz–Manna extension as well as with the transitive closure of the one-step Dershowitz–Manna extension, allowing users to switch between the three characterizations. Remarkably, the characterizations do not coincide for arbitrary orders. Each characterization has its advantages. For example, the Huet–Oppen extension is unsuitable for defining the recursive path order [10, Section 2], whereas the Dershowitz–Manna extension is unsuitable for defining the Knuth–Bendix order [5, Section 2].

▶ **Example 1** (McCarthy's 91 Function)**.** Dershowitz and Manna [15] apply the multiset order to prove the termination of a tail-recursive reformulation of McCarthy's 91 function. Using Isabelle's definitional mechanism for recursive functions [23], the 91 function is specified as

function $\mathsf{g} : \mathit{nat} \to \mathit{int} \to \mathit{int}$ **where**
$\mathsf{g}\ n\ z = (\text{if } n = 0 \text{ then } z \text{ else if } z > 100 \text{ then } \mathsf{g}\ (n-1)\ (z-10) \text{ else } \mathsf{g}\ (n+1)\ (z+11))$

To perform this definition, the command needs a well-founded relation R that includes g's call graph; otherwise, we could define a function f such that $\mathsf{f}\ n = \mathsf{f}\ n + 1$ and use this to derive $0 = 1$, a contradiction. The default automation [24, 12] is not powerful enough to synthesize the relation, so we provide R ourselves. Following Dershowitz and Manna (but correcting '< 111' to '≤ 111'), we define the relation as

$$R = \big\{((n, z), (n', z'))).\ (\tau\ n\ z,\ \tau\ n'\ z') \in \mathsf{mult}\ \{(a, b).\ b < a \wedge a \leq 111\}\big\}$$

where τ is defined as follows, together with an auxiliary function f:

definition $\mathsf{f} : \mathit{int} \to \mathit{int}$ **where** $\mathsf{f}\ x = (\text{if } x > 100 \text{ then } x - 10 \text{ else } 91)$
definition $\tau : \mathit{nat} \to \mathit{int} \to \mathit{int}\ \mathit{multiset}$ **where** $\tau\ n\ z = \mathsf{mset}\ (\mathsf{map}\ (\lambda i.\ \mathsf{f}^i\ z)\ [0\,..\,n-1])$

The mset function in τ's definition converts a list to a multiset of its elements.

The main proof obligation is to show that g's call graph is included in R – i.e., that the arguments (n, z) become smaller with each recursive call according to the measure $\tau\ n\ z$ and the multiset order. We followed the original proof, relying on existing lemmas about mult.

The verified SAT solver framework [7] developed mainly by Fleury as part of the IsaFoL (Isabelle Formalization of Logic) effort represents clauses as multisets of literals and problems as multisets of clauses. To improve automation, we developed simplifier plugins, or "simprocs," that cancel terms that appear on both sides of a subtraction, equality, or inequality, rewriting the expression $A + \{x\} + B = \mathsf{add}\ x\ A$ to $B = \{\}$ and $A + \{x\} + B - \mathsf{add}\ x\ A$ to B. Since multiset multiplicities are natural numbers, we started with the cancellation simprocs for *nat*, due to Lawrence Paulson, and generalized them to arbitrary members of the type class of cancellative commutative monoids – including *nat*, *'a multiset*, and the hereditary and signed multiset types introduced in Sections 5 and 7. The properties required by the type class are $0 + a = a$, $(a + b) + c = a + (b + c)$, $a + b = b + a$, $(a + b) - a = b$, and $a - b - c = a - (b + c)$.

The simprocs for *nat* depend on a '$k\ \times$' operation. We first needed to define its multiset counterpart, $\mathsf{repeat} : \mathit{nat} \to \mathit{'a\ multiset} \to \mathit{'a\ multiset}$, and to prove lemmas about it. The *multiset* instances of our simprocs collectively perform the following steps:

1. Normalize the goal by rewriting $\mathsf{add}\ a\ X$ to $\{a\} + X$ and $\mathsf{replicate}\ n\ a$ to $\mathsf{repeat}\ n\ \{a\}$.
2. Extract A and B from any occurrence of the pattern $A \sim B$ in the goal to normalize, where $\sim$ is among $-$, $=$, $<$, $\leq$, $\subset$, and $\subseteq$.
3. Extract the summands in $A = A_1 + \cdots + A_m$ and $B = B_1 + \cdots + B_n$ to form two lists of multiplicity–term pairs.
4. Find common terms on both sides, subtract the coefficients, and remove the element in the goal using an explicit lemma instantiation.
5. Recombine the simplified terms with $\sim$.
6. Normalize $\{a\}$ back to $\mathsf{add}\ a\ \{\}$ and simplify add (e.g., replacing $M + \mathsf{add}\ a\ \{\} + N$ with $\mathsf{add}\ a\ (M + N)$).

In general, the normalization steps are parameterized by rewrite rules, which must be provided for each type instance.

4 Nested Multisets

The type of nested multisets defined in Section 1 is freely generated by the constructors $\mathsf{Elem} : \mathit{'a} \to \mathit{'a\ nmultiset}$ and $\mathsf{MSet} : (\mathit{'a\ nmultiset})\ \mathit{multiset} \to \mathit{'a\ nmultiset}$. The characteristic theorems derived by the **datatype** command [8] include the induction rule

$$(\bigwedge x.\ P\ (\mathsf{Elem}\ x)) \Longrightarrow (\bigwedge M.\ (\bigwedge N \in NM.\ P\ N) \Longrightarrow P\ (\mathsf{MSet}\ NM)) \Longrightarrow P\ N\,.$$

In the $\mathsf{MSet}\ NM$ case, the induction hypothesis applies to all elements N of the multiset NM. Strictly speaking, the condition is $N \in \mathsf{set}\ NM$, where set is the natural transformation associated with the functor *multiset*, but we overload the symbol $\in$ for multisets. The command also defines a recursor $\mathsf{rec} : (\mathit{'a} \to \mathit{'b}) \to ((\mathit{'a\ nmultiset} \times \mathit{'b})\ \mathit{multiset} \to \mathit{'b}) \to \mathit{'a\ nmultiset} \to \mathit{'b}$ and derives its characteristic equations:

$$\mathsf{rec}\ e\ ms\ (\mathsf{Elem}\ x) = e\ x$$
$$\mathsf{rec}\ e\ ms\ (\mathsf{MSet}\ NM) = ms\ (\mathsf{image}\ (\lambda N.\ (N, \mathsf{rec}\ e\ ms\ N))\ NM)$$

Using the recursor, we can specify primitive recursive functions on nested multisets. A useful example is the depth of a nested multiset, $\mathsf{depth} : 'a\ \mathit{nmultiset} \to \mathit{nat}$, defined by

$$\mathsf{depth} = \mathsf{rec}\ (\lambda x.\ 0)\ (\lambda M.\ \mathrm{let}\ X = \mathsf{set}\ (\mathsf{image}\ \mathsf{snd}\ M)\ \mathrm{in\ if}\ X = \{\}\ \mathrm{then}\ 0\ \mathrm{else}\ \mathsf{Max}\ X + 1)$$

where $\mathsf{snd} : 'a \times 'b \to 'b$ is the second pair projection and Max returns the maximum element of a nonempty finite set equipped with a linear order. Even for a simple example like this, the recursor-based definition is cryptic. Isabelle's **primrec** command allows users to specify primitive recursive functions by their equations, such as

$$\begin{aligned}\mathsf{depth}\ (\mathsf{Elem}\ x) &= 0\\ \mathsf{depth}\ (\mathsf{MSet}\ M) &= (\mathrm{let}\ X = \mathsf{set}\ (\mathsf{image}\ \mathsf{depth}\ M)\ \mathrm{in\ if}\ X = \{\}\ \mathrm{then}\ 0\ \mathrm{else}\ \mathsf{Max}\ X + 1)\end{aligned}$$

Internally, **primrec** defines depth in terms of rec and derives the user's equations from rec's characteristic equations.

The next function we consider is Dershowitz and Manna's nested multiset order [15]. We reproduce their definition below (partly adapting the notations):

For two nested multisets M, N over A, we say that $M \ll^* N$ if
1. $M, N \in A$ and $M < N$ (two elements of the base set are compared using $<$); or else
2. $M \notin A$ and $N \in A$ (any multiset is greater than any element of the base set); or else
3. $M, N \notin A$, and for some nested multisets X, Y, where $\{\} \neq Y \subseteq N$,

$$M = (N - Y) \uplus X \quad \text{and} \quad \forall x \in X.\ \exists y \in Y.\ x \ll^* y\,.$$

The corresponding Isabelle definition is as follows (with $<$ overloaded to also mean $\ll^*$):

$$\begin{aligned}&\textbf{function}\ < : 'a\ \mathit{nmultiset} \to 'a\ \mathit{nmultiset} \to \mathit{bool}\ \textbf{where}\\ &\quad \mathsf{Elem}\ a < \mathsf{Elem}\ b \longleftrightarrow a < b\\ &\mid \mathsf{Elem}\ a < \mathsf{MSet}\ M \longleftrightarrow \mathsf{True}\\ &\mid \mathsf{MSet}\ M < \mathsf{Elem}\ a \longleftrightarrow \mathsf{False}\\ &\mid \mathsf{MSet}\ M < \mathsf{MSet}\ N \longleftrightarrow \mathsf{ext_{DM}}\ (<)\ M\ N\end{aligned}$$

There are several things to note here. First, we use **function** instead of **primrec** because we are recursing on two nested multisets simultaneously. Second, $\mathsf{ext_{DM}}\ R\ A\ B$ is the Dershowitz–Manna multiset order extension of R applied to multisets A, B. It is defined by

$$\mathsf{ext_{DM}}\ R\ A\ B \longleftrightarrow \exists X\ Y.\ Y \neq \{\} \wedge Y \subseteq B \wedge A = (B - Y) + X \wedge \forall x \in X.\ \exists y \in Y.\ R\ x\ y\,.$$

Third, the **function** command expects a termination proof in the form of a well-founded relation. We provide the well-founded lexicographic product $\mathsf{sub} \times_{\mathsf{lex}} \mathsf{sub}$ of the immediate subterm relations $\mathsf{sub} : (\mathit{nmultiset} \times \mathit{nmultiset})\ \mathit{set}$ defined as $\{(N, \mathsf{MSet}\ NM) \mid N \in NM\}$. The termination proof crucially relies on the fact that $\mathsf{ext_{DM}}$ applies the relation passed as its first argument only to nested multisets contained in its second and third arguments. This is easy to prove for arbitrary relations for $\mathsf{ext_{DM}}$ but would be harder for mult. And since we have not established the transitivity of the function we are defining yet, we cannot use the equivalence of mult and $\mathsf{ext_{DM}}$ on partial orders. This explains the use $\mathsf{ext_{DM}}$ in the definition. Fourth, after obtaining the termination proof, **function** generates an induction principle that matches the recursion schema used in the definition.

Next, we prove several closure properties of the nested multiset order. If $< : 'a \to 'a \to \mathit{bool}$ is a (nonstrict) preorder, then $< : 'a\ \mathit{nmultiset} \to 'a\ \mathit{nmultiset} \to \mathit{bool}$ yields a preorder. The same closure property holds for partial orders, total orders, and wellorders (i.e., well-founded

total orders). Each closure property corresponds to a type class instantiation, and each type class instantiation gives us a wealth of lemmas and helpful reasoning infrastructure about the nested multiset order.

Only the proofs of transitivity and well-foundedness of the nested multiset order are challenging. For transitivity, we rely on $\mathsf{ext}_{\mathsf{DM}}$'s equivalence to mult on transitive relations. We prove transitivity by induction, and the induction hypothesis establishes that $\mathsf{ext}_{\mathsf{DM}}$ is only applied to a transitive relation. To prove well-foundedness, we start with an auxiliary lemma: $<$ is well-founded on the set of nested multisets of a fixed depth i. Formally: wf $\{(M, N) \mid \mathsf{depth}\ M = i \wedge \mathsf{depth}\ N = i \wedge M < N\}$. The proof proceeds by induction on i and uses the equivalence of $\mathsf{ext}_{\mathsf{DM}}$ and mult (on transitive relations) as well as the proof of preservation of well-foundedness by mult. Finally, we obtain the well-foundedness of the entire relation by observing that depth $M <$ depth N implies $M < N$ and hence $M < N$ can be rewritten to depth $M <$ depth $N \vee$ depth $M =$ depth $N \wedge M < N$. In other words, we lexicographically compare the depths and resort to the nested multiset order to break ties. The claim follows, since with the above lemma both components of the lexicographic comparison are well founded.

5 Hereditary Multisets

Many authors rely on nested multisets and the order on them in one way or another. However, most of them do not use the Elem constructor, or use it in an easily avoidable way. In this section, we consider nested multisets with no Elem constructor. In set theory, we could simply let $'a$ be the empty set to model this. Since this is not possible in HOL, we define Elem-freedom as an inductive predicate:

inductive no_elem : $'a$ *nmultiset* $\to$ *bool* **where**
$(\bigwedge N \in NM.\ \mathsf{no\_elem}\ N) \Longrightarrow \mathsf{no\_elem}\ (\mathsf{MSet}\ NM)$

In principle, we could now use **typedef** to carve out a new type consisting of nested multisets satisfying no_elem. However, the resulting type would be isomorphic to the datatype of hereditary multisets as introduced in Section 1, with its single injective constructor HMSet : *hmultiset multiset* $\to$ *hmultiset*. We prefer the datatype definition, since it offers convenient recursion and induction schemas. Nonetheless, the subtype view on hereditary multisets is also useful, as it allows us to lift the infrastructure from nested to hereditary multisets.

Formally, we establish this view by providing an isomorphism, as two mutually inverse injections Abs : *unit nmultiset* $\to$ *hmultiset* and Rep : *hmultiset* $\to$ *unit nmultiset* defined by Abs (MSet M) = HMSet (image Abs M) and Rep (HMSet M) = MSet (image Rep M). The isomorphism follows by easy inductions (on *hmultiset* or on the definition of no_elem):

lemma $\bigwedge M$: *hmultiset*. no_elem (Rep M)
$\bigwedge M$: *hmultiset*. Abs (Rep M) = M
$\bigwedge N$: *unit nmultiset*. no_elem $N \Longrightarrow$ Rep (Abs N) = N

The Lifting tool [20] exploits these properties to lift constants on nested multisets to hereditary multisets. Here is our definition of the hereditary multiset order:

lift_definition $<$: *hmultiset* $\to$ *hmultiset* $\to$ *bool* **is**
$<$: *unit nmultiset* $\to$ *unit nmultiset* $\to$ *bool*

The **lift_definition** command produces the definition $M < N \leftrightarrow \mathsf{Rep}\ M < \mathsf{Rep}\ N$, which hides the isomorphism. Moreover, the command also provides a setup for the companion Transfer tool [20], which reduces proof goals about the abstract type (*hmultiset*) to goals about the raw type (*unit nmultiset*). Nevertheless, the proof of the expected property $\mathsf{HMSet}\ M < \mathsf{HMSet}\ N \leftrightarrow \mathsf{ext}_{\mathsf{DM}}\ (<)\ M\ N$ is not trivial, because we must ensure that the witnesses for the existential quantifiers in the definition of $\mathsf{ext}_{\mathsf{DM}}$ contain no Elem. With this property in place, it is easy to lift the instantiations of the various order type classes (up to and including wellorders) from nested to hereditary multisets.

Finally, we prove that *hmultiset* is a cancellative commutative monoid by lifting the corresponding structure from the *multiset* type – i.e., by defining $0 = \mathsf{HMSet}\ \{\}$, $\mathsf{HMSet}\ A + \mathsf{HMSet}\ B = \mathsf{HMSet}\ (A + B)$, and $\mathsf{HMSet}\ A - \mathsf{HMSet}\ B = \mathsf{HMSet}\ (A - B)$. This enables natural-number-like reasoning with multisets, including our generalized cancellation simprocs.

6 Syntactic Ordinals

Hereditary multisets are isomorphic to the ordinals below ϵ_0, which we call the *syntactic ordinals*. Instead of defining a new type, we use *hmultiset* whenever we need such ordinals and provide the main ordinal operations on this type. Our notion of syntactic ordinal is similar to Dershowitz and Moser's "ordinal terms" [16], which they attribute to Takeuti [35, Section 2.11], but unlike them we do not need to consider the ordinal 0 specially.

The empty hereditary multiset 0 corresponds to the ordinal 0. Union + corresponds to Hessenberg addition, which is traditionally denoted by $\oplus$; we write + to exploit the Isabelle type classes for addition. The multiset order $<$ conveniently coincides with its ordinal counterpart. Multiset subtraction also makes sense as a truncating subtraction on ordinals; for example, $1 - 3 = 0$, $\omega - 3 = \omega$, and $\omega^2 + 3 - \omega = \omega^2 + 3$. Notice that if $\alpha < \beta$, then it is not necessarily the case that $\alpha - \beta = 0$. To provide more complete support for ordinals, we define some additional constants and abbreviations on *hmultiset*, starting with the following basic concepts: $\omega^\alpha = \mathsf{HMSet}\ \{\alpha\}$; $1 = \omega^0$; $\omega = \omega^1$. Given 1 and +, Isabelle lets us enter arbitrary numerals and interprets them as $1 + \cdots + 1$.

The Hessenberg product is defined by taking the cartesian product of the operands' multisets and applying addition on the resulting pairs to obtain a multiset of ordinals, i.e., an ordinal: $\alpha \cdot \beta = \mathsf{HMSet}\ (\mathsf{image}\ (+)\ (\mathsf{hmsetmset}\ \alpha \times \mathsf{hmsetmset}\ \beta))$, where $\mathsf{hmsetmset}$ is HMSet's inverse. The expected properties are easy to prove: Multiplication is associative, commutative, and distributive over addition; 0 is absorbent; 1 is neutral; $0 \neq 1$; etc.

It is interesting to compare our definition of the Hessenberg product with the literature. The following definition, by Ludwig and Waldmann [26], illustrates how suboptimal abstractions can lead to convoluted formulations:

- For $\alpha \in \mathbf{O} \setminus \{0\}$ we define: $\quad 0 \odot 0 = 0, \quad 0 \odot \alpha = 0, \quad \alpha \odot 0 = 0.$
- Let for $m, m' \in \mathbb{N}^{>0}$, $n_1, \ldots, n_m$, $n'_1, \ldots, n'_{m'} \in \mathbb{N}^{>0}$, $b_1, \ldots, b_m$, $b'_1, \ldots, b'_{m'} \in \mathbf{O}$ such that $b_1 > b_2 > \cdots > b_m$ and $b'_1 > b'_2 > \cdots > b'_m$,

  $$a = \textstyle\sum_{i=1}^{m} (\omega^{b_i} \cdot n_i),\ b = \sum_{i=1}^{m'} (\omega^{b'_i} \cdot n'_i)\,.$$

  We define then

  $$\alpha \odot \beta = \textstyle\bigoplus_{i=1}^{m} \bigoplus_{j=1}^{m'} \left(\omega^{b_i \oplus b'_j} \cdot \left(\mathrm{coeff}(\alpha, b_i) + \mathrm{coeff}(\beta, b'_j)\right)\right)\,.$$

Subtraction is so ill behaved that multiplication does not distribute over it: For $\alpha = \omega^2 + \omega$, $\beta = 1$, and $\gamma = \omega$, we have $\alpha \cdot (\beta - \gamma) = \omega^2 + \omega \neq \omega = \alpha \cdot \beta - \alpha \cdot \gamma$. As a result, some Isabelle type classes for subtraction cannot be instantiated for ordinals.

Given an ordinal $\sum_{i=1}^{n}(\omega^{\alpha_i} \cdot c_i)$ in Cantor normal form, its degree corresponds to the largest exponent, α_1. Unfortunately, this definition does not gracefully handle the case where $n = 0$. We could extend the ordinal type with a special value (e.g., $-\infty$), but this would require a different type and operations on that type. Instead, we introduce the concept of a *head ω-factor*. For 0, the head ω-factor is 0; for nonzero ordinals of degree α, it is ω^α, which is always nonzero. The degree is the maximum element in the multiset that corresponds to the ordinal. Formally: $\mathsf{head}_\omega\ \alpha = \big(\text{if } \alpha = 0 \text{ then } 0 \text{ else } \omega^{\mathsf{Max}\ (\mathsf{set}\ (\mathsf{hmsetmset}\ \alpha))}\big)$.

The following decomposition lemma, which was brought to our attention by Uwe Waldmann, is useful when comparing ordinals. Given two ordinals α_1, α_2 such that $\alpha_1 < \alpha_2$, we can always express them as sums of the form $\alpha_i = \gamma + \beta_i$, where the head ω-factor of β_1 is smaller than that of β_2:

lemma *hmset_pair_decompose_less*:
$\alpha_1 < \alpha_2 \Longrightarrow \exists\gamma\ \beta_1\ \beta_2.\ \alpha_1 = \gamma + \beta_1 \wedge \alpha_2 = \gamma + \beta_2 \wedge \mathsf{head}_\omega\ \beta_1 < \mathsf{head}_\omega\ \beta_2$

▶ **Example 2** (Hydra Battle)**.** The hydra battle [22] is a classic process whose termination cannot be proved by induction on the natural numbers. Following Dershowitz and Moser's reformulation [16], we use Lisp-style lists (unlabeled binary trees) to represent hydras: **datatype** *lisp* = Nil | Cons *lisp lisp*. The functions car and cdr are defined to extract the first and second arguments of Cons, respectively; they return Nil when invoked on a Nil node.

A hydra consists of a list of heads. In a departure from standard Greek mythology, each head is recursively a list of heads. The battle also involves a hero, Hercules, who gets to chop off one of the hydra's "leaf" heads at each round. If the head has no grandparent node, the head is lost and there is no regrowth; otherwise, the branch issuing from the grandparent node is copied n times, where n starts at 1 and is increased by 1 in each round. We refer to the literature for more details (and some nice drawings of polycephalic monsters).

Formally, the battle is captured by the h function below, which depends on an auxiliary function d:

function d : *nat* → *lisp* → *lisp* **where**
d $n\ x$ = (if car x = Nil then cdr x
else if car (car x) = Nil then (Cons (cdr (car x)))^n (cdr x)
else Cons (d n (car x)) (cdr x))

function h : *nat* → *lisp* → *lisp* **where**
h $n\ x$ = (if x = Nil then Nil else h $(n + 1)$ (d $n\ x$))

For d, termination is easy: The left subtree of x becomes smaller with each iteration, so we can take $\{((n', x'), (n, x)).\ |x'| < |x|\}$ as the relation, where $|x|$ returns the number of nodes of a list x. For h, instead of $|x|$, we use encode x as the measure, where encode is defined by the primitive recursive equations encode Nil = 0 and encode (Cons $l\ r$) = $\omega^{\mathsf{encode}\ l}$ + encode r.

Thanks to well-foundedness of $<$, it suffices to show that the ordinal decreases in each recursive call to h – i.e., if $x \neq$ Nil, then encode (d $n\ x$) $<$ encode x. The proof is by structural induction on x. The Nil case is trivial. In the Cons $l\ r$ case, if l = Nil, we have $0 < \omega^0 +$ encode r. Otherwise, we distinguish two subcases. If car l = Nil, we must prove encode (f n (cdr l) r) $< \omega^{\mathsf{encode}\ l}$ + encode r, which amounts to $n \cdot \omega^{\mathsf{encode}\ (\mathsf{cdr}\ l)} +$ encode $r < \omega^{\omega^{\mathsf{encode}\ (\mathsf{car}\ l)} + \mathsf{encode}\ (\mathsf{cdr}\ l)} +$ encode r. The right-hand side is greater because the exponent to ω has an additional nonzero term, $\omega^{\mathsf{encode}\ (\mathsf{car}\ l)}$, which dwarfs the n factor on the left. In the remaining case, we have car $l \neq$ Nil. The proof obligation amounts to encode (d $n\ l$) $<$ encode l, corresponding to the induction hypothesis for the left subtree.

The termination proof is about 30 lines long in Isabelle. As is often the case, the formalization revealed some inaccuracies in the informal argument. Although this is not

mentioned by Dershowitz and Moser, the termination proof depends on $\mathsf{car}\ \mathsf{Nil} = \mathsf{Nil}$. They also claim that $\mathsf{Cons}\ \mathsf{Nil}\ \mathsf{Nil}$ represents a "leaf of a hydra," but the $\mathsf{car}\ x = \mathsf{Nil}$ test in d's definition can only mean that a simple Nil node is used for leaves.

▶ **Example 3** (Ludwig and Waldmann)**.** The following Isar proof outline closely follows the pen-and-paper proof of a property Ludwig and Waldmann needed to reason about their transfinite Knuth–Bendix order [26]. The informal proof was communicated to us privately by Waldmann. A more restrictive formulation is claimed as Lemma 10 in their paper. The steps are justified by short proofs, omitted below, most of which were generated by Sledgehammer. Using Isabelle's built-in automation and the cancellation simprocs, it would be possible to eliminate some of the intermediate steps.

lemma
  assumes $\alpha_2 + \beta_2 \cdot \gamma < \alpha_1 + \beta_1 \cdot \gamma$ **and** $\beta_2 \le \beta_1$ **and** $\gamma < \delta$
  shows $\alpha_2 + \beta_2 \cdot \delta < \alpha_1 + \beta_1 \cdot \delta$
proof
  obtain $\eta\ \gamma'\ \delta'$ **where** $\gamma = \eta + \gamma'$ **and** $\delta = \eta + \delta'$ **and** $\mathsf{head}_\omega\ \gamma' < \mathsf{head}_\omega\ \delta'$
  obtain $\beta_0\ \beta_1'\ \beta_2'$ **where** $\beta_1 = \beta_0 + \beta_1'$ **and** $\beta_2 = \beta_0 + \beta_2'$ **and**
    $\mathsf{head}_\omega\ \beta_2' < \mathsf{head}_\omega\ \beta_1' \vee \beta_2' = 0 \wedge \beta_1' = 0$ {by *hmset_pair_decompose_less*}

  have $\alpha_2 + \beta_0 \cdot \gamma + \beta_2' \cdot \gamma = \alpha_2 + \beta_2 \cdot \gamma$ {by $\beta_2 = \beta_0 + \beta_2'$}
  also have $< \alpha_1 + \beta_1 \cdot \gamma$ {by $\alpha_2 + \beta_2 \cdot \gamma < \alpha_1 + \beta_1 \cdot \gamma$}
  also have $= \alpha_1 + \beta_0 \cdot \gamma + \beta_1' \cdot \gamma$ {by $\beta_1 = \beta_0 + \beta_1'$}
  finally have $\alpha_2 + \beta_2' \cdot \gamma < \alpha_1 + \beta_1' \cdot \gamma$ {by cancellation}

  have $\alpha_2 + \beta_2 \cdot \delta = \alpha_2 + \beta_0 \cdot \delta + \beta_2' \cdot \delta$ {by $\beta_2 = \beta_0 + \beta_2'$}
  also have $= \alpha_2 + \beta_0 \cdot \delta + \beta_2' \cdot \eta + \beta_2' \cdot \delta'$ {by $\delta = \eta + \delta'$}
  also have $\le \alpha_2 + \beta_0 \cdot \delta + \beta_2' \cdot \eta + \beta_2' \cdot \delta' + \beta_2' \cdot \gamma'$ {by monotonicity}
  also have $= \alpha_2 + \beta_2' \cdot \gamma + \beta_0 \cdot \delta + \beta_2' \cdot \delta'$ {by $\gamma = \eta + \gamma'$}
  also have $< \alpha_1 + \beta_1' \cdot \gamma + \beta_0 \cdot \delta + \beta_2' \cdot \delta'$ {by $\alpha_2 + \beta_2' \cdot \gamma < \alpha_1 + \beta_1' \cdot \gamma$}
  also have $= \alpha_1 + \beta_1' \cdot \eta + \beta_1' \cdot \gamma' + \beta_0 \cdot \eta + \beta_0 \cdot \delta' + \beta_2' \cdot \delta'$ {by $\gamma = \eta + \gamma'$, $\delta = \eta + \delta'$}
  also have $\le \alpha_1 + \beta_1' \cdot \eta + \beta_0 \cdot \eta + \beta_0 \cdot \delta' + \beta_1' \cdot \delta'$
    {by $\mathsf{head}_\omega\ (\beta_1' \cdot \gamma' + \beta_2' \cdot \delta') < \mathsf{head}_\omega\ (\beta_1' \cdot \delta') \vee \beta_1' \cdot \gamma' = \beta_2' \cdot \delta' = \beta_1' \cdot \delta' = 0$}
  finally show $\alpha_2 + \beta_2 \cdot \delta < \alpha_1 + \beta_1 \cdot \delta$ {by $\beta_1 = \beta_0 + \beta_1'$, $\delta = \eta + \delta'$}
qed

7 Signed Variants of Multisets and Hereditary Multisets

Syntactic ordinals do not enjoy the property that Ludwig and Waldmann refer to as "continuity" [26]: Given syntactic ordinals α, β such that $\alpha < \beta$, there does not necessarily exist an ordinal γ such that $\alpha + \gamma = \beta$. Yet, syntactic ordinals correspond formally to polynomials over the indeterminate ω. Why not allow negative coefficients (e.g., $\omega^2 - 2\omega + 1$) and take $\gamma = \beta - \alpha$? The practical motivation arose in the context of the formalization of a transfinite Knuth–Bendix order [4, 5]. Although it is a simple idea, we could not find it in the literature.

We start by defining the *signed multisets*, also called the hybrid multisets [2]. In principle, we could define them in a similar way as the plain multisets (Section 3), by substituting *int* for *nat* in the **typedef** command. However, we prefer to perform a quotient construction, so as to be able to lift operations and lemmas about plain multisets:

quotient_type *'a zmultiset* = *'a multiset* × *'a multiset* / *R*

where $R = \lambda(P_1, N_1)\ (P_2, N_2).\ P_1 + N_2 = P_2 + N_1$. The **quotient_type** command [21] introduces a type that is isomorphic to the set of R-equivalence classes. In the same way as a pair (m, n) of natural numbers can represent the integer $m - n$, we use a pair (P, N) of plain multisets to capture the signed multiset $P - N$, whose multiplicities can be negative. For example, both $(\{\}, \{7\})$ and $(\{3\}, \{3, 7\})$ correspond to the signed multiset that contains 7 with multiplicity -1 and no other element. Notice that $R\ (\{\}, \{7\})\ (\{3\}, \{3, 7\})$, since $\{\} + \{3, 7\} = \{7\} + \{3\}$. The pos and neg functions, of type *'a zmultiset* $\to$ *'a multiset*, return normalized P and N components, with $P \cap N = \{\}$.

The main signed multiset operations are defined by specifying them on the raw level of multiset pairs and lifting them to the abstract level of equivalence classes. The Lifting tool emits a proof obligation stating that equivalence classes are respected. For example, for unary functions f on multisets, this means that when f is invoked on R-equivalent arguments x and y, the results $f\ x$ and $f\ y$ are R-equivalent. A few definitions are given below:

lift_definition $0 :$ *'a zmultiset* **is** $(\{\}, \{\})$

lift_definition $-$ $:$ *'a zmultiset* $\to$ *'a zmultiset* $\to$ *'a zmultiset* **is**
$\lambda(P_1, N_1)\ (P_2, N_2).\ (P_1 + N_2,\ N_1 + P_2)$

lift_definition zcount $:$ *'a zmultiset* $\to$ *'a* $\to$ *int* **is**
$\lambda(P, N)\ x.$ int (count $P\ x$) $-$ int (count $N\ x$)

Many lemmas about such definitions can be lifted from the raw types using the Transfer tool, which exploits relational parametricity [31] to transfer results across types. Most HOL functions arising in practice are parametric.

The type *zhmultiset* of signed hereditary multisets is defined using the **typedef** command as isomorphic to *hmultiset zmultiset*. Notice that the multisets contained in such a signed multiset are not signed. It is unclear to us whether "hereditarily signed" multisets, and "ordinals" in which the exponents of ω can recursively contain negative coefficients, would be worth studying.

The ι function embeds *hmultiset* into *zhmultiset*. Operations such as $+$, $-$, and $<$ are lifted from the underlying multisets. Ordinal multiplication is by far the most problematic operation. It can be defined in terms of the cartesian product on signed multisets:

lift_definition $\cdot$ $:$ *zhmultiset* $\to$ *zhmultiset* $\to$ *zhmultiset* **is**
$\lambda M\ N$: *hmultiset zmultiset*.
$\quad \iota$ (hmsetmset (HMSet (pos M) $\cdot$ HMSet (pos N)))
$- \iota$ (hmsetmset (HMSet (pos M) $\cdot$ HMSet (neg N)))
$+ \iota$ (hmsetmset (HMSet (neg M) $\cdot$ HMSet (neg N)))
$- \iota$ (hmsetmset (HMSet (neg M) $\cdot$ HMSet (pos N)))

It is difficult to prove the associativity of this multiplication operator. Using the Transfer tool, we can quickly reduce the proof obligation to the following property of unsigned ordinals:

$$\begin{aligned}
&\alpha_2 \cdot (\beta_2 \cdot \gamma_2 + \beta_1 \cdot \gamma_1 - (\beta_2 \cdot \gamma_1 + \gamma_2 \cdot \beta_1)) + \gamma_2 \cdot (\alpha_2 \cdot \beta_1 + \beta_2 \cdot \alpha_1 - (\alpha_2 \cdot \beta_2 + \alpha_1 \cdot \beta_1)) + \\
&\alpha_1 \cdot (\beta_2 \cdot \gamma_1 + \gamma_2 \cdot \beta_1 - (\beta_2 \cdot \gamma_2 + \beta_1 \cdot \gamma_1)) + \gamma_1 \cdot (\alpha_2 \cdot \beta_2 + \alpha_1 \cdot \beta_1 - (\alpha_2 \cdot \beta_1 + \beta_2 \cdot \alpha_1)) \\
= \ &\alpha_2 \cdot (\beta_2 \cdot \gamma_1 + \gamma_2 \cdot \beta_1 - (\beta_2 \cdot \gamma_2 + \beta_1 \cdot \gamma_1)) + \gamma_2 \cdot (\alpha_2 \cdot \beta_2 + \alpha_1 \cdot \beta_1 - (\alpha_2 \cdot \beta_1 + \beta_2 \cdot \alpha_1)) + \\
&\alpha_1 \cdot (\beta_2 \cdot \gamma_2 + \beta_1 \cdot \gamma_1 - (\beta_2 \cdot \gamma_1 + \gamma_2 \cdot \beta_1)) + \gamma_1 \cdot (\alpha_2 \cdot \beta_1 + \beta_2 \cdot \alpha_1 - (\alpha_2 \cdot \beta_2 + \alpha_1 \cdot \beta_1))
\end{aligned}$$

After staring at this goal for a few hours, we speculated that the property $\alpha \cdot (\gamma - \beta) + \alpha \cdot \beta = \alpha \cdot (\beta - \gamma) + \alpha \cdot \gamma$ holds about truncating subtraction. We proved it and applied it four times to make the left-hand side of the proof obligation above syntactically identical to the right-hand side.

Here is a selection of the properties we proved using Isabelle, with α, β, γ ranging over signed ordinals:

1. $\alpha + \beta = \beta + \alpha$;
2. $(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma)$;
3. $\alpha \cdot \beta = \beta \cdot \alpha$;
4. $(\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma)$;
5. $(\alpha + \beta) \cdot \gamma = \alpha \cdot \gamma + \beta \cdot \gamma$;
6. $\alpha < \beta \leftrightarrow \alpha + \gamma < \beta + \gamma$;
7. $0 + \alpha = \alpha$;
8. $0 \cdot \alpha = 0$;
9. $1 \cdot \alpha = \alpha$;
10. $\alpha \leq \beta \leftrightarrow \alpha < \beta + 1$;
11. $\alpha \cdot \beta = 0 \leftrightarrow \alpha = 0 \vee \beta = 0$;
12. $\alpha < \beta \wedge 0 < \gamma \Longrightarrow \gamma \cdot \alpha < \gamma \cdot \beta$;
13. $\alpha - \beta + \beta = \alpha$;
14. $\alpha - \beta + \gamma = \alpha + \gamma - \beta$;
15. $\alpha + \beta - \gamma = \alpha + (\beta - \gamma)$;
16. $\alpha - \beta - \gamma = \alpha - (\beta + \gamma)$.

Finally, we instantiated our generalized cancellation simprocs for *zmultiset* and *zhmultiset*. As preprocessing steps, we normalize unary minuses, rewriting $-\alpha + \beta + \alpha$ to $(\beta + \alpha) - \alpha$; the subtraction simproc then cancels the two α's, yielding β.

▶ **Example 4** (Ludwig and Waldmann, Continued)**.** By exploiting signed ordinals, we obtain a much simpler proof of Ludwig and Waldmann's property from Example 3:

have $\iota\,\alpha_2 + \iota\,\beta_2 \cdot \iota\,\delta = \iota\,\alpha_2 + \iota\,\beta_2 \cdot \iota\,\gamma + \iota\,\beta_2 \cdot (\iota\,\delta - \iota\,\gamma)$	{by algebraic manipulations}
also have $< \iota\,\alpha_1 + \iota\,\beta_1 \cdot \iota\,\gamma + \iota\,\beta_2 \cdot (\iota\,\delta - \iota\,\gamma)$	{by $\alpha_2 + \beta_2 \cdot \gamma < \alpha_1 + \beta_1 \cdot \gamma$}
also have $\leq \iota\,\alpha_1 + \iota\,\beta_1 \cdot \iota\,\gamma + \iota\,\beta_1 \cdot (\iota\,\delta - \iota\,\gamma)$	{by $\beta_2 \leq \beta_1$, $\gamma < \delta$}
also have $= \iota\,\alpha_1 + \iota\,\beta_1 \cdot \iota\,\delta$	{by algebraic manipulations}
finally show $\alpha_2 + \beta_2 \cdot \delta < \alpha_1 + \beta_1 \cdot \delta$	{by algebraic manipulations}

The next-to-last step eliminates the subtraction $\iota\,\delta - \iota\,\gamma$, paving the way for the final step, which removes the ι casts. Waldmann privately confirmed that he was aware of this shortcut but did not dare take it without a solid theoretical foundation for signed ordinals.

8 Application to Goodstein's Theorem

Goodstein's theorem [17] states that every Goodstein sequence eventually terminates at 0. Before we can define these sequences, we must first introduce an auxiliary notion. A natural number is in *hereditary base* n if it is expressed as a product $c_1 n^k + c_2 n^{k-1} + \cdots + c_{k-1} n + c_k$, where $0 \leq c_i < n$ for each i, $c_1 \neq 0$, and the exponents $k, k-1, \ldots, 1$ are recursively expressed in hereditary base n. For example, 500 is written as $2 \cdot 3^{3+2} + 3^2 + 3 + 2$ in hereditary base 3.

The Goodstein sequence $\mathcal{G}_s$ of natural numbers is defined as follows. The starting value is given by s: $\mathcal{G}_s(0) = s$. The remaining values $\mathcal{G}_s(i+1)$ are obtained by expressing $\mathcal{G}_s(i)$ in base $i+2$, replacing all occurrences of $i+2$ by $i+3$, and subtracting 1. For example, we have $\mathcal{G}_4(0) = 4 = 2^2$, $\mathcal{G}_4(1) = 3^3 - 1 = 26 = 2 \cdot 3^2 + 2 \cdot 3 + 2$, and $\mathcal{G}_4(2) = 2 \cdot 4^2 + 2 \cdot 4 + 2 - 1 = 41$. Somewhat counterintuitively, the sequence eventually converges to 0: $\mathcal{G}_4(3 \cdot 2^{402\,653\,211} - 1) = 0$.

Our formal proof relies on two functions encode and decode that convert between natural numbers and hereditary base n. The adjective 'hereditary' suggests that hereditary multisets, or syntactic ordinals, are a suitable data structure. Following this idea, $2 \cdot \omega^2 + 2 \cdot \omega + 2$ represents $2 \cdot 3^2 + 2 \cdot 3 + 2$ in hereditary base 3, or $2 \cdot 5^2 + 2 \cdot 5 + 2$ in hereditary base 5.

The encode and decode functions are defined in a local context that fixes a constant base ≥ 2. Beyond the end of the local context, we must supply the base explicitly as additional argument to encode and decode, which we will indicate as a subscript.

context
  fixes base : *nat*
  assumes base ≥ 2
begin

function encode : *nat* $\to$ *nat* $\to$ *hmultiset* **where**
  encode e n = (if $n = 0$ then 0 else (n mod base) $\cdot$ $\omega^{\text{encode } 0\ e}$ + encode $(e+1)$ (n/base))

primrec decode : *nat* $\to$ *hmultiset* $\to$ *nat* **where**
  decode e (HMSet M) = $\left(\sum_{\alpha \in M} \text{base}^{\text{decode } 0\ \alpha}\right) / \text{base}^e$

end

The argument e gives the exponent of the current base, starting from 0. The recursion scheme for encode is nonprimitive. Termination is established using the measure $\lambda(e, n).\ n \cdot (\text{base}^e + 1)$. The Goodstein sequence is defined as follows, where start is fixed:

primrec goodstein : *nat* $\to$ *nat* **where**
  goodstein 0 = start
| goodstein $(i+1)$ = $\text{decode}_{i+3}\ 0\ (\text{encode}_{i+2}\ 0\ (\text{goodstein}\ i)) - 1$

If goodstein $i > 0$, then the ordinal associated with goodstein $(i+1)$ is smaller than the one for goodstein i. Let $\mathcal{E}_i = \text{encode}_{i+2}\ 0\ (\text{goodstein}\ i)$. The proof sketch is as follows:

lemma *goodstein_step*:
  assumes goodstein $i > 0$
  shows $\mathcal{E}_i > \mathcal{E}_{i+1}$
proof
  have $\text{decode}_{i+3}\ 0\ \mathcal{E}_i > 0$ {by *decode_0_iff*}
  have $\mathcal{E}_i = \text{encode}_{i+3}\ 0\ (\text{decode}_{i+3}\ 0\ \mathcal{E}_i)$ {by *encode_decode_exp_0*}
  also have $> \text{encode}_{i+3}\ 0\ (\text{decode}_{i+3}\ 0\ \mathcal{E}_i - 1)$
    {by *less_imp_encode_less*, $\text{decode}_{i+3}\ 0\ \mathcal{E}_i > 0$}
  finally show $\mathcal{E}_i > \mathcal{E}_{i+1}$ {by definition of $\mathcal{E}_{i+1}$}
qed

The main lemmas that are directly or indirectly needed are listed below:

lemma *less_imp_encode_less*: $n < p \Longrightarrow \text{encode}\ e\ n < \text{encode}\ e\ p$
lemma *less_imp_decode_less*:
  well_base $\alpha \Longrightarrow$ aligned $e\ \alpha \Longrightarrow$ aligned $e\ \beta \Longrightarrow \alpha < \beta \Longrightarrow \text{decode}\ e\ \alpha < \text{decode}\ e\ \beta$
lemma *decode_0_iff*: well_base $\alpha \Longrightarrow$ aligned $e\ \alpha \Longrightarrow (\text{decode}\ e\ \alpha = 0 \longleftrightarrow \alpha = 0)$
lemma *decode_encode*: decode e (encode $e\ n$) $= n$
lemma *encode_decode_exp_0*: well_base $\alpha \Longrightarrow$ encode 0 (decode $0\ \alpha$) $= \alpha$

The first and second properties are the most difficult ones. The first one is proved by well-founded induction, following the recursion structure of encode. This induction principle is derived automatically by the **function** command. The second property is proved by strong induction on α. The assumptions ensure that the coefficients stored in α are smaller than the base (well_base) and that the last e digits are all 0s (aligned).

The entire formalization is about 580 lines long. The main difficulty was to come up with the right lemmas and inductions. Reasoning about ordinals was fairly comfortable. Nevertheless, we are very impressed by Zankl, Winkler, and Middeldorp's automatic proof of the termination of a term rewriting system that computes Goodstein's sequence [40]. Possibly

the key to their success is that they avoid converting back and forth between the natural numbers and hereditary base notation.

9 Application to Decidability of Unary PCF

Plotkin's PCF language of "programming computable functions" [30] is a simply typed λ-calculus that has natural numbers $\mathbb{N}$ as a base type and permits recursion on them. Types are interpreted as Scott domains, i.e., $[\![\mathbb{N}]\!] = \mathbb{N} \cup \{\bot\}$. For unary PCF, a fragment of PCF which has only the base type o with the single value $\top$ (in addition to the domain's $\bot$), behavioral equivalence is decidable [25, 32]. Schmidt-Schauß's elegant proof [32] is based on an inductive enumeration of representative terms. The termination of the enumeration is ensured by abstracting types into hereditary multisets. In our ongoing formalization effort of this decidability result (which poses many challenges unrelated to multisets), we proved Schmidt-Schauß's key Lemma 11 about hereditary multisets.

Types of unary PCF are defined as **datatype** $\mathit{type} = \mathsf{o} \mid \mathit{type} \Rightarrow \mathit{type}$, where $\Rightarrow$ is a right-associative infix datatype constructor. Given a type T, we define its argument types T_i and its arity $\mathsf{ar}\ T$ such that $T = T_0 \Rightarrow \cdots \Rightarrow T_{\mathsf{ar}\ T - 1} \Rightarrow \mathsf{o}$. We measure a type using a primitive recursive function $\delta : \mathit{type} \to \mathit{hmultiset}$ defined by $\delta\ \mathsf{o} = 0$ and $\delta\ (T \Rightarrow U) = \omega^{\delta\ T} + \delta\ U$. For a type T, Schmidt-Schauß constructs a set of representative closed terms of behavioral equivalence classes. The construction is recursive and relies on a decrease in the involved types' measures for termination. More precisely, given T, the construction recursively computes sets of representative terms for types $\pi_i^j\ T$ for all $i < \mathsf{ar}\ T$ and $j \le \mathsf{ar}\ T_i$, where the operator π is defined recursively as follows:

fun $\pi : \mathit{nat} \to \mathit{nat} \to \mathit{type} \to \mathit{type}$ **where**
  $\pi_i^0\ T = (\text{if } i < \mathsf{ar}\ T \text{ then } T_0 \Rightarrow \cdots \Rightarrow T_{i-1} \Rightarrow T_{i+1} \Rightarrow \cdots \Rightarrow T_{\mathsf{ar}\ T-1} \Rightarrow \mathsf{o} \text{ else } \mathsf{o})$
| $\pi_i^{j+1}\ T = (\text{if } i < \mathsf{ar}\ T \wedge j < \mathsf{ar}\ T_i \text{ then } \pi_j^0\ T_i \Rightarrow \cdots \Rightarrow \pi_j^{\mathsf{ar}\ (T_i)_j}\ T_i \Rightarrow \pi_i^0\ T \text{ else } \mathsf{o})$

Finally, we prove that π indeed decreases the measure of types using the induction principle that follows the structure of π's definition and is provided by Isabelle's **fun** command [23].

lemma δ_π:
  assumes $i < \mathsf{ar}\ T$ **and** $j \le \mathsf{ar}\ T_i$
  shows $\delta\ (\pi_i^j\ T) < \delta\ T$
proof (*induct rule*: $\pi.\mathit{induct}$)
  fix $T\ i$
  assume $i < \mathsf{ar}\ T$
  show $\delta\ (\pi_i^0\ T) < \delta\ T$ {by definition of δ and π and simple multiset reasoning}
next
  fix $T\ i\ j$
  assume $i < \mathsf{ar}\ T$ **and** $j < \mathsf{ar}\ T_i$ **and**
    IH_1: $\delta\ (\pi_j^0\ T_i) < \delta\ T_i$ **and** IH_2: $\forall k < \mathsf{ar}\ (T_i)_j.\ \delta\ (\pi_j^{k+1}\ T_i) < \delta\ T_i$
  define $\mathsf{X} = \{\delta\ (\pi_j^0\ T_i)\} + \mathsf{image}\ (\lambda k.\ \delta\ (\pi_j^{k+1}\ T_i))\ \{0, \ldots, \mathsf{ar}\ (T_i)_j - 1\}$ **and**
    $\mathsf{Y} = \{\delta\ T_i\}$ **and** $\mathsf{Z} = \mathsf{image}\ \delta\ \{T_0, \ldots, T_{i-1}, T_{i+1}, \ldots, T_{\mathsf{ar}\ T-1}\}$
  have $\delta\ (\pi_i^{j+1}\ T) = \mathsf{HMSet}\ (\mathsf{X} + \mathsf{Z})$ {by definitions of δ and π}
  also have $\mathsf{X} + \mathsf{Z} < \mathsf{Y} + \mathsf{Z}$ {by Dershowitz–Manna characterization of $<$, IH_1, IH_2}
  also have $\mathsf{HMSet}\ (\mathsf{Y} + \mathsf{Z}) = \delta\ T$ {by definition of δ}
  finally show $\delta\ (\pi_i^{j+1}\ T) < \delta\ T$ {by above calculation}
qed

The key step to help automation is to define X, Y, and Z (all of type *hmultiset multiset*) such that after unfolding the Dershowitz–Manna definition of the multiset order, the inequality is easily fulfilled (given the induction hypotheses).

Our formal proof is very close to Schmidt-Schauß's informal argument. It is rare to be able to formalize a technical proof so closely. This can be due to the care taken by the informal proof writer, due to good formal library support, or due to a combination of both.

10 Related Work

The nonnested multiset order has been formalized in several proof assistants, including Coq [11, 14], HOL4, and Isabelle/HOL [36]. The Isabelle version is generalized to take two parameters: a strict and a nonstrict order. The resulting order is strictly more powerful for termination proving than the standard version. Another aspect where the related work goes further than our work is executability. In principle, little stands in the way of a decent code generation setup for our multiset variants, following the lines of the existing multiset setup.

Norrish and Huffman [28] present two formalizations of ordinals, in HOL4 and Isabelle/HOL. The HOL4 formalization models ordinals as quotients of wellorders with respect to wellorder isomorphism. The Isabelle/HOL development also relies on a quotient construction, but from a more syntactic notion of raw ordinals, defined as **datatype** *preordinal* = Zero | StrictLim (*nat* → *preordinal*). Independently, Blanchette, Popescu, and Traytel [9] formalized ordinals and cardinals in Isabelle/HOL, representing ordinals by well-ordered relations dispersed over arbitrary types. Thereby they avoided fixing an a priori bound on the ordinals that can be constructed. All these formalizations go beyond ϵ_0 but are ultimately limited by the expressiveness of HOL, which is strictly weaker than set theory. Another difference with our current work is that they provide the standard addition and product and not Hessenberg's.

In Coq, Castéran and Contejean [13] formalized ordinal notations up to Γ_0. As case studies, they considered the hydra battle and Goodstein's theorem. Grimm [18] ported and extended this work, covering three alternative notions of ordinals. Also in Coq, Vermaat [38] formalized tree ordinals, a syntactic representation similar to Norrish and Huffman's. Recently, Schmitt [33] axiomatized ordinals in KeY and proved Goodstein's theorem.

11 Conclusion

We presented a formalization in Isabelle/HOL of nested multisets, hereditary multisets, and ordinals below ϵ_0. Their datatype definitions emphasize the close connections between the three notions. The signed generalizations of these types, with potentially negative multiplicities, offer a subtraction operator that enjoys nice algebraic properties. The signed syntactic ordinals do not appear to have been studied before.

The Lifting and Transfer tools were invaluable for carrying definitions across different multiset types. We relied heavily on Sledgehammer; it sometimes generated complex Isar proofs, which we occasionally inserted in our development. The cancellation simprocs also played a role, after we had adapted them so that they work on multisets. Well-founded recursion using **function** was vital, as it often is. But perhaps the most noteworthy aspect of our work is that all the necessary types could be introduced easily, either as inductive datatypes, as subsets of existing types, or as quotients. The support for recursion through bounded natural functors, a distinguishing feature of Isabelle/HOL, was crucial to define nested and hereditary multisets in a simple way, with convenient induction and recursion schemas.

Acknowledgment. Lawrence Paulson gave us the idea to formalize the nested multiset order. Aart Middeldorp pointed us to related work. Uwe Waldmann shared his insights about syntactic ordinals with us, including the lemma and the proof of Example 3. Robert Lewis, Mark Summerfield, Sophie Tourret, and the (extremely thorough) anonymous reviewers suggested dozens of textual improvements.

References

1 Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, pages 19–99. Elsevier, 2001.

2 Jean-Pierre Banâtre, Pascal Fradet, and Yann Radenac. Generalised multisets for chemical programming. *Math. Struct. Comput. Sci.*, 16(4):557–580, 2006.

3 Gilles Barthe, Venanzio Capretta, and Olivier Pons. Setoids in type theory. *J. Funct. Program.*, 13(2):261–293, 2003.

4 Heiko Becker, Jasmin Christian Blanchette, Uwe Waldmann, and Daniel Wand. Formalization of Knuth–Bendix orders for lambda-free higher-order terms. *Archive of Formal Proofs*, 2016. URL: `https://devel.isa-afp.org/entries/Lambda_Free_KBOs.shtml`.

5 Heiko Becker, Jasmin Christian Blanchette, Uwe Waldmann, and Daniel Wand. A transfinite Knuth–Bendix order for lambda-free higher-order terms. In Leonardo de Moura, editor, *CADE-26*, LNCS. Springer, 2017.

6 Jasmin Christian Blanchette, Mathias Fleury, and Dmitriy Traytel. Formalization of nested multisets, hereditary multisets, and syntactic ordinals. *Archive of Formal Proofs*, 2016. URL: `https://devel.isa-afp.org/entries/Nested_Multisets_Ordinals.shtml`.

7 Jasmin Christian Blanchette, Mathias Fleury, and Christoph Weidenbach. A verified SAT solver framework with learn, forget, restart, and incrementality. In Nicola Olivetti and Ashish Tiwari, editors, *IJCAR 2016*, volume 9706 of *LNCS*, pages 25–44. Springer, 2016.

8 Jasmin Christian Blanchette, Johannes Hölzl, Andreas Lochbihler, Lorenz Panny, Andrei Popescu, and Dmitriy Traytel. Truly modular (co)datatypes for Isabelle/HOL. In Gerwin Klein and Ruben Gamboa, editors, *ITP 2014*, volume 8558 of *LNCS*, pages 93–110. Springer, 2014.

9 Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Cardinals in Isabelle/HOL. In Gerwin Klein and Ruben Gamboa, editors, *ITP 2014*, volume 8558 of *LNCS*, pages 111–127. Springer, 2014.

10 Jasmin Christian Blanchette, Uwe Waldmann, and Daniel Wand. A lambda-free higher-order recursive path order. In Javier Esparza and Andrzej Murawski, editors, *FoSSaCS 2017*, volume 10203 of *LNCS*, pages 461–479. Springer, 2017.

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

12 Lukas Bulwahn, Alexander Krauss, and Tobias Nipkow. Finding lexicographic orders for termination proofs in Isabelle/HOL. In Klaus Schneider and Jens Brandt, editors, *TPHOLs 2007*, volume 4732 of *LNCS*, pages 38–53. Springer, 2007.

13 Pierre Castéran and Évelyne Contejean. On ordinal notations. *Coq User Contributions*, 2006. URL: `http://www.lix.polytechnique.fr/coq/V8.2pl1/contribs/Cantor.html`.

14 Evelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons, and Xavier Urbain. Certification of automated termination proofs. In Boris Konev and Frank Wolter, editors, *FroCoS 2007*, volume 4720 of *LNCS*, pages 148–162. Springer, 2007.

15 Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Commun. ACM*, 22(8):465–476, 1979.

16 Nachum Dershowitz and Georg Moser. The Hydra Battle revisited. In Hubert Comon-Lundh, Claude Kirchner, and Hélène Kirchner, editors, *Rewriting, Computation and Proof, Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of His 60th Birthday*, volume 4600 of *LNCS*, pages 1–27. Springer, 2007.

17 R. L. Goodstein. On the restricted ordinal theorem. *J. Symb. Logic*, 9(2):33–41, 1944.

18 José Grimm. Implementation of three types of ordinals in Coq. Technical Report RR-8407, Inria, 2013. URL: `https://hal.inria.fr/hal-00911710/document`.

19 Gerard Huet and Derek C. Oppen. Equations and rewrite rules: A survey. In Ronald V. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405. Academic Press, 1980.

20 Brian Huffman and Ondřej Kunčar. Lifting and Transfer: A modular design for quotients in Isabelle/HOL. In Georges Gonthier and Michael Norrish, editors, *CPP 2013*, volume 8307 of *LNCS*, pages 131–146. Springer, 2013.

21 Cezary Kaliszyk and Christian Urban. Quotients revisited for Isabelle/HOL. In William C. Chu, W. Eric Wong, Mathew J. Palakal, and Chih-Cheng Hung, editors, *SAC 2011*, pages 1639–1644. ACM, 2011.

22 Laurie Kirby and Jeff Paris. Accessible independence results for Peano arithmetic. *Bull. London Math. Soc.*, 4:285–293, 1982.

23 Alexander Krauss. Partial recursive functions in higher-order logic. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR 2006*, volume 4130 of *LNCS*, pages 589–603. Springer, 2006.

24 Alexander Krauss. Certified size-change termination. In Frank Pfenning, editor, *CADE-21*, volume 4603 of *LNCS*, pages 460–475. Springer, 2007.

25 Ralph Loader. Unary PCF is decidable. *Theor. Comput. Sci.*, 206(1-2):317–329, 1998.

26 Michel Ludwig and Uwe Waldmann. An extension of the Knuth-Bendix ordering with LPO-like properties. In Nachum Dershowitz and Andrei Voronkov, editors, *LPAR 2007*, volume 4790 of *LNCS*, pages 348–362. Springer, 2007.

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

28 Michael Norrish and Brian Huffman. Ordinals in HOL: Transfinite arithmetic up to (and beyond) ω_1. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *ITP 2013*, volume 7998 of *LNCS*, pages 133–146. Springer, 2013.

29 Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *IWIL-2010*, volume 2 of *EPiC*, pages 1–11. EasyChair, 2012.

30 Gordon D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977.

31 John C. Reynolds. Types, abstraction and parametric polymorphism. In *IFIP'83*, pages 513–523, 1983.

32 Manfred Schmidt-Schauß. Decidability of behavioural equivalence in unary PCF. *Theor. Comput. Sci.*, 216(1-2):363–373, 1999.

33 Peter H. Schmitt. A first-order theory of ordinals. In Cláudia Nalon and Renate Schmidt, editors, *TABLEAUX 2017*, LNCS. Springer, 2017.

34 Wacław Sierpiński. *Cardinal and Ordinal Numbers*, volume 34 of *Polska Akademia Nauk Monografie Matematyczne*. Państwowe Wydawnictwo Naukowe, 1958.

35 Gaisi Takeuti. *Proof Theory*, volume 81 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2nd edition, 1987.

36 René Thiemann, Guillaume Allais, and Julian Nagele. On the formalization of termination techniques based on multiset orderings. In Ashish Tiwari, editor, *RTA 2012*, volume 15 of *LIPIcs*, pages 339–354. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.

37 Dmitriy Traytel, Andrei Popescu, and Jasmin Christian Blanchette. Foundational, compositional (co)datatypes for higher-order logic: Category theory applied to theorem proving. In *LICS 2012*, pages 596–605. IEEE Computer Society, 2012.

38 Martijn Vermaat. *Infinitary Rewriting in Coq*. M.Sc. thesis, Vrije Universiteit Amsterdam, 2010. URL: `http://www.cs.vu.nl/~tcs/mt/vermaat.pdf`.

39 Makarius Wenzel. Isabelle/Isar – A generic framework for human-readable proof documents. In Roman Matuszewski and Anna Zalewska, editors, *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, volume 10(23) of *Studies in Logic, Grammar, and Rhetoric*. Uniwersytet w Białymstoku, 2007.

40 Harald Zankl, Sarah Winkler, and Aart Middeldorp. Beyond polynomials and Peano arithmetic – Automation of elementary and ordinal interpretations. *J. Symb. Comput.*, 69:129–158, 2015.

41 Hans Zantema. Termination. In Marc Bezem, Jan Willem Klop, and Roel de Vrijer, editors, *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*, pages 181–259. Cambridge University Press, 2003.

Observably Deterministic Concurrent Strategies and Intensional Full Abstraction for Parallel-or*

Simon Castellan[1], Pierre Clairambault[2], and Glynn Winskel[3]

1 **Université Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France**
Simon.Castellan@ens-lyon.fr
2 **Université Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France**
Pierre.Clairambault@ens-lyon.fr
3 **Computer Laboratory, University of Cambridge, Cambridge, UK**
Glynn.Winskel@cl.cam.ac.uk

Abstract

Although Plotkin's *parallel-or* is inherently deterministic, it has a non-deterministic interpretation in games based on (prime) event structures – in which an event has a unique causal history – because they do not directly support disjunctive causality. *General event structures* can express disjunctive causality and have a more permissive notion of determinism, but do not support hiding. We show that (structures equivalent to) *deterministic* general event structures do support hiding, and construct a new category of games based on them with a deterministic interpretation of $\mathbf{aPCF}_{\mathtt{por}}$, an affine variant of **PCF** extended with parallel-or. We then exploit this deterministic interpretation to give a relaxed notion of determinism (observable determinism) on the plain event structures model. Putting this together with our previously introduced concurrent notions of well-bracketing and innocence, we obtain an intensionally fully abstract model of $\mathbf{aPCF}_{\mathtt{por}}$.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases Game semantics, parallel-or, concurrent games, event structures, full abstraction

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.12

1 Introduction

Plotkin's *parallel-or* is, in a sense, the most well-understood of programming language primitives. Recall that it is a primitive $\mathsf{por} : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ defined by the three equations:

$$\mathsf{por}\ \mathtt{tt}\ \bot = \mathtt{tt}\,, \qquad \mathsf{por}\ \bot\ \mathtt{tt} = \mathtt{tt}\,, \qquad \mathsf{por}\ \mathtt{ff}\ \mathtt{ff} = \mathtt{ff}\,.$$

As Plotkin famously proved [16], it is the primitive to be added to the paradigmatic purely functional higher-order programming language **PCF** in order to get a perfect match (*full abstraction*) with respect to Scott domains. Since Plotkin's result, full abstraction has been the gold standard for semanticists. Indeed, whereas an adequate model is always *sound* to reason about program equivalence; a *fully abstract model* is also *complete*: two programs are observationally equivalent if and only if they have the same denotation. Though since Plotkin's paper fully abstract models of various programming languages have been proposed (often through game semantics, with *e.g.* state [3, 1], control [5, 14], concurrency [11],

* This work was partially supported by French LABEX MILYON (ANR-10-LABX-0070), and by the ERC Advanced Grant ECSYM.

...); they are usually quite a bit more complicated than plain Scott domains. Parallel-or is well-understood, in that it has a simple input-output (*extensional*) behaviour, and its presence in **PCF** reduces observational equivalence to input-output behaviour.

But is it *really* so well-understood? While parallel-or is simple *extensionally*, how is it best understood *intensionally*? What is the operational behaviour of $\mathsf{por}\, M\, N$? Any programmer will sense the subtleties in implementing such a primitive. It immediately spawns two threads, starting evaluating M and N *in parallel*. If both terminate with ff, it terminates with ff as well after both have finished. But it suffices that *one* of the threads returns tt, for parallel-or to return tt immediately, discarding the other. If *both* return tt then one of the threads will be discarded, but which one depends on the scheduler. From an input-output perspective it does not matter which one is selected as they race to produce the same result, but the race nonetheless happens from an operational viewpoint; and a sufficiently intensional semantics will show it. Such behaviour is useful in practice, for instance to speed up deciding the existence of a "good" branch in a search tree by spawning a thread for each branch to explore.

From a game semantics perspective, understanding this combination of racy concurrency and deterministic extensional behaviour has been an unexpected challenge. In earlier work, we dealt with deterministic parallelism using games and strategies based on event structures [17], showing an intensional full abstraction result for a parallel implementation of **PCF** [8]. We also showed how our tools supported shared memory concurrency [6]. Given that, it is no surprise that the racy behaviour mentioned above can be represented with event structures, yielding an adequate model of *e.g.* **PCF** plus parallel-or. But game semantics based on event structures is *causal*: each event comes with a unique causal history. Dependency is *conjunctive*: an event can only occur after *all* its dependencies. This feature is key for the notion of concurrent innocence leading to our definability result [8] and in fact for the very construction of our basic category of games [7]. This has the unfortunate consequence of forcing the strategy for parallel-or to be *non-deterministic*: the race in the operational behaviour of $\mathsf{por}\ \mathsf{tt}\ \mathsf{tt}$ yields a choice between two events competing to return tt, each depending on one argument of por. But interpreting por through non-determinism really is a workaround, as appears through the resulting failure of full abstraction.

Constructing a deterministic intensional model for parallel-or has proved demanding. Plays-based models [11] fail as they inline the non-deterministic choice of the scheduler, and cannot express even pure parallelism deterministically. We saw above that plain event structure games do not work either. Giving a deterministic model for parallel-or involves modifying the latter to allow *disjunctive causality*: a given event may have several distinct causal histories, and an event occurrence does not carry information on its specific causal history. Alternatives to event structures allowing this have existed for a long time: *general event structures* [18]. However, we will see that they do not, in general, support *hiding* – which is required to build a category of strategies. We will show that they do support it modulo some further conditions, making them adequate to model parallel-or deterministically.

Beyond the historical twist of giving a fully abstract games model for a language with parallel-or (as game semantics was originally driven by the full abstraction problem for **PCF** *without* parallel-or [13, 2]), a treatment of disjunctive causality is indispensable in a complete game semantics framework for concurrency. The following example illustrates how mundane and widespread it is: the causal history of a packet arriving from the network is simply its *route*. Clearly, introducing an event for each route is best avoided if possible – especially since all further events depending on it will be hereditarily duplicated as well. Deterministic models for disjunctive causality serve two purposes: they allow for (1) a more general, less intensional notion of determinism, and (2) a coarser equivalence relation between strategies

that abstracts away benign races. These issues, that we address in this paper, have become recurrent obstacles in our research programme; and the historical importance of parallel-or in semantics made it the perfect candidate to test and showcase the solution.

Contributions. Concretely, we build an intensionally fully abstract games model of $\mathbf{aPCF}_{\mathtt{por}}$, an *affine* version of **PCF** with parallel-or. Our tools are designed with the extension in the presence of *symmetry* [8] to full $\mathbf{PCF}_{\mathtt{por}}$ in mind; but presenting them in an affine case allows us to focus on the issues pertaining to parallel-or and disjunctive causality, orthogonal to symmetry. Presenting everything at once in conference format is not reasonable.

Full abstraction for $\mathbf{aPCF}_{\mathtt{por}}$ has two facets: on the one hand, we need to import the conditions of innocence and well-bracketing from [8], which rely on *conjunctive* causality. On the other hand, we need a disjunctive notion of determinism. Consequently our model will involve: (1) the standard category **CG** of concurrent games on event structures, and (2) the main novelty of our paper, a new category **Disj** supporting disjunctive determinism. Glueing the two together, we can import in **CG** the notion of determinism from **Disj**, thereafter dubbed *observational determinism*, and prove intensional full abstraction.

Related work. *Game semantics* is a branch of denotational semantics. Originally driven by the full abstraction problem for **PCF**, it has grown in the past 25 years into a powerful and versatile methodology to construct compositionally intensional representations of program execution. This lead on the one hand to many full abstraction results (particularly striking in the presence of state as the fully abstract models are then effectively presentable [3, 1]), and on the other hand to applications, ranging from program analysis (model-checking or equivalence checking) to compilation and hardware synthesis [10].

Within game semantics, the present paper is part of a line of work on so-called "truly concurrent" game semantics first pushed by Melliès and colleagues [4, 15], and which advocates the use of causal structures such as event structures and asynchronous transition systems in interactive semantics of programming languages. This line of work has seen a lot of activity in the past five years, prompted in part by a new category of games and strategies on event structures generalizing all prior work, introduced by Rideau and the last author [17]; but also the presheaf-based framework for concurrent games by Hirschowitz and colleagues [12].

Finally, our account of disjunctive causality relies heavily on determinism for *hiding* to work. The third author and Marc de Visme have developed *event structures with disjunctive causality (edc)* [9], supporting both disjunctive causality and hiding in a non-deterministic setting. Links between these two approaches are being explored; at the very least, extra axioms would have to be imposed on edcs in order to mimic the work here.

Outline. In Section 2 we introduce $\mathbf{aPCF}_{\mathtt{por}}$ and give its (non-deterministic) event structure games model **CG**. In Section 3 we introduce the deterministic model **Disj** for parallel-or. Finally in Section 4 we glue the two, generalize the conditions of [8] and prove full abstraction.

2 Causal game semantics for affine PCF with parallel-or

2.1 The language $\mathrm{aPCF}_{\mathtt{por}}$

To alleviate notation, we only take booleans $\mathbb{B}$ as base type. We start with **affine PCF** (**aPCF**). Its types are either $\mathbb{B}$, or $A \multimap B$ for some types A and B. Its terms are those of the

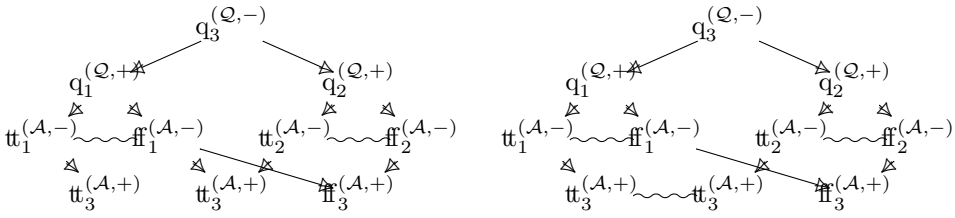


Figure 1 Two strategies on $\mathbb{B}_1 \multimap \mathbb{B}_2 \multimap \mathbb{B}_3$, for parallel-lor and parallel-or.

affine λ-calculus with a divergence $\bot$ and boolean primitives (constants and conditionals):

$$M, N \quad ::= \quad \lambda x.\, M \mid M\, N \mid x \mid \mathtt{tt} \mid \mathtt{ff} \mid \text{if } M \text{ then } N_1 \text{ else } N_2 \mid \bot$$

We skip the affine typing rules, which are standard – the typing of if M then N_1 else N_2 is additive, *i.e.* N_1 and N_2 may share resources, and the N_i have boolean type. The (call-by-name) operational semantics, also standard, yield an evaluation relation $M \Downarrow v$ between closed terms and *values* (booleans or abstractions). We write $M \Downarrow$ when there is some v such that $M \Downarrow v$, and $M \Uparrow$ otherwise. Two terms $\Gamma \vdash M, N : A$ are **observationally equivalent** ($M \simeq_{\text{obs}} N$) iff for all contexts $\mathcal{C}[-]$ such that $\mathcal{C}[M], \mathcal{C}[N]$ are closed terms of type $\mathbb{B}$, $\mathcal{C}[M] \Downarrow \Leftrightarrow \mathcal{C}[N] \Downarrow$. Recall that an interpretation $[\![-]\!]$ in some model $\mathcal{M}$ is **fully abstract** if for all M, N, $M \simeq_{\text{obs}} N$ iff $[\![M]\!] = [\![N]\!]$ – it is **intensionally fully abstract** if $\mathcal{M}$ quotiented by the semantic equivalent of $\simeq_{\text{obs}}$ is fully abstract.

In [16], Plotkin proved that Scott domains are fully abstract not for **PCF**, but for its extension with the so-called **parallel-or** operation:

$$\frac{M \Downarrow \mathtt{tt}}{\mathsf{por}\, M\, N \Downarrow \mathtt{tt}}, \qquad \frac{N \Downarrow \mathtt{tt}}{\mathsf{por}\, M\, N \Downarrow \mathtt{tt}}, \qquad \frac{M \Downarrow \mathtt{ff} \qquad N \Downarrow \mathtt{ff}}{\mathsf{por}\, M\, N \Downarrow \mathtt{ff}}.$$

The combinator por is *not sequential*: it is not the case that it evaluates one of its arguments first. In particular, $\mathsf{por}\, \mathtt{tt}\, \bot \Downarrow \mathtt{tt}$ and $\mathsf{por}\, \bot\, \mathtt{tt} \Downarrow \mathtt{tt}$.

2.2 Strategies as event structures

As usual in game semantics, computation is a dialogue between the program and its environment. The *moves*, or *events*, are either due to the program (Player, $+$) or its environment (Opponent, $-$). They are either variable calls (Questions, $\mathcal{Q}$) or returns (Answers, $\mathcal{A}$). Unlike traditional game semantics, in our line of work such dialogues are *partially ordered.*

Figure 1 presents two concurrent strategies, both playing on (the game for) $\mathbb{B}_1 \multimap \mathbb{B}_2 \multimap \mathbb{B}_3$, where subscripts are for disambiguation. The diagrams are read from top to bottom. The first event in both strategies is an Opponent question on $\mathbb{B}_3$, initiating the computation. Then Player (the program) starts evaluating in parallel its two arguments (q_1 and q_2). These may return $\mathtt{tt}$ or $\mathtt{ff}$, the wiggly line indicating that they cannot reply *both*. Depending on these, Player may eventually answer q_3. In both diagrams, $\mathtt{ff}_3$ requires *both* arguments to evaluate to $\mathtt{ff}$; however they differ as to the events that trigger an answer $\mathtt{tt}_3$. These diagrams will be made more formal later, but we invite the readers to examine them and convince themselves that the first diagram represents a parallel implementation of the *left or* (diverging if its first argument diverges), whereas the second diagram represents *parallel or*.

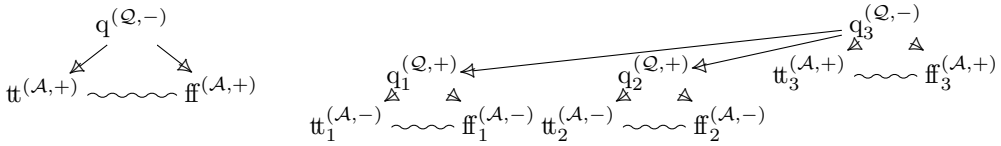


Figure 2 Representations of the arenas $[\![\mathbb{B}]\!]$ and $[\![\mathbb{B}_1 \multimap \mathbb{B}_2 \multimap \mathbb{B}_3]\!]$.

Event structures. Such diagrams are formalized as *event structures*. We use here event structures with binary conflict, whereas those of [17, 7] have a more general set of *consistent sets*. Binary conflict is sufficient for our purposes, and preserved by all operations we need.

▶ **Definition 1.** A (prime) **event structure** (with binary conflict, **es** for short) is $(E, \leq_E, \#_E)$ where E is a set of events, $\leq_E$ is a partial order on E called **causality** and $\#_E$ is an irreflexive symmetric binary relation called *conflict*, such that:

- $\forall e \in E,\ [e] = \{e' \in E \mid e' \leq_E e\}$ is finite,
- $\forall e \# e',\ \forall e' \leq_E e'',\ e \# e''$.

We will often omit the indices in $\leq_E$, $\#_E$ if they are obvious from the context.

In the second axiom, we say that the conflict (e, e'') is **inherited** from (e, e'). If a conflict (e, e') is not inherited (meaning dependencies of e and e' are pairwise **compatible**, *i.e.* non-conflicting), we say that it is a **minimal conflict** and denote it by $e \sim\!\!\sim e'$. The *states* of an event structure E, called **configurations**, are the finite sets $x \subseteq E$ that are both consistent (events are pairwise compatible) and **down-closed** (*i.e.* for all $e \in x$, for all $e' \leq e$, then $e' \in x$) – the set of configurations on E is written $\mathscr{C}(E)$, and is partially ordered by inclusion. Configurations with a maximal element are called **prime configurations**, they are those of the form $[e]$ for $e \in E$. We will also use the notation $[e) = [e] \setminus \{e\}$. Between configurations, the **covering relation** $x \mathrel{-\!\!\subset} y$ means that y is obtained from x by adding exactly one event: y is an **atomic extension** of x. We might also write $x \overset{e}{\mathrel{-\!\!\subset}}$ to mean that $e \notin x$ and $x \cup \{e\} \in \mathscr{C}(E)$. Finally, when drawing event structures, we will not represent the full partial order $\leq$ but the **immediate causality** generating it, defined as $e \rightarrowtriangle e'$ whenever $e < e'$ and for any $e \leq e'' \leq e'$, either $e = e''$ or $e'' = e'$. The diagrams of Figure 1 represent event structures; only displaying immediate causality $\rightarrowtriangle$ and minimal conflict $\sim\!\!\sim$.

Arenas. Besides causality and conflict, events in Figure 1 carry *labels* $(\mathrm{q}, \mathrm{tt}, \mathrm{ff}, \dots)$: formally, those will come from the *game*, or the *arena*. Arenas are the semantic representatives of types. They are certain event structures with polarities and Questions/Answers labeling.

▶ **Definition 2.** An **arena** is a triple $(A, \mathsf{pol}_A, \lambda_A)$ where A is an event structure, $\mathsf{pol}_A : A \to \{-, +\}$ and $\lambda_A : A \to \{\mathcal{Q}, \mathcal{A}\}$ are labelings for polarity and Questions/Answers, such that:

- The order $\leq_A$ is *forest-shaped*: for $a_1, a_2 \leq a \in A$, either $a_1 \leq a_2$ or $a_2 \leq a_1$.
- The relation $\rightarrowtriangle_A$ is *alternating*: if $a_1 \rightarrowtriangle_A a_2$, then $\mathsf{pol}_A(a_1) \neq \mathsf{pol}(a_2)$.
- If $\lambda_A(a_2) = \mathcal{A}$, then there is $a_1 \rightarrowtriangle a_2$, and $\lambda_A(a_1) = \mathcal{Q}$,
- A is *race-free*: if $a_1 \sim\!\!\sim a_2$, then $\mathsf{pol}_A(a_1) = \mathsf{pol}_A(a_2)$.

An arena A is **negative** if all its minimal events have negative polarity.

Conflict aside, our arenas resemble those of [13]: the justification relation traditionally denoted by $\vdash_A$ is simply $\rightarrowtriangle_A$. Basic arenas include the empty arena 1, and the arena $[\![\mathbb{B}]\!]$ displayed in Figure 2, often written $\mathbb{B}$ by abuse of notation, for booleans.

We mention here some constructions on arenas. The **dual** $A^\perp$ of A is obtained by taking $\mathsf{pol}_{A^\perp} = -\mathsf{pol}_A$, and leaving the rest unchanged. The **simple parallel composition** $A \parallel B$ is obtained as having events the tagged disjoint union $\{1\} \times A \cup \{2\} \times B$, and all components inherited. The **product** $A \,\&\, B$ of negative A and B is obtained as $A \parallel B$, with all events of A in conflict with events of B. As types will be denoted by *negative* arenas, we need a negative arena to interpret $\multimap$. This is done by setting A to depend on (negative) minimal events of B. If B has at most one minimal event, this is easy:

▶ **Definition 3.** Let A, B be negative arenas. Assume that B is **well-opened**, *i.e.* $\min(B)$ has at most one event. If $B = 1$, then $A \multimap B = 1$. Otherwise, $\min(B) = \{b_0\}$. We define $A \multimap B$ as $A^\perp \parallel B$, with the additional causal dependency $(2, b_0) \leq (1, a)$ for all $a \in A$.

Defining $A \multimap B$ for well-opened B is sufficient to interpret $\mathbf{aPCF}_{\mathtt{por}}$ types, but would be insufficient in the presence of a tensor type; with *e.g.* $\mathbb{B} \multimap (\mathbb{B} \otimes \mathbb{B})$. The complication here is due to a *disjunctive causality* at the level of types: the left $\mathbb{B}$ is caused by either of the occurrences of $\mathbb{B}$ on the right. As for parallel-or, the inability of event structures to express that can be worked around by introducing two conflicting copies of the left $\mathbb{B}$, one for each causal justification – this is done in [6], and is reminiscent of the standard arena construction of [13]. This works well for some purposes, but the informed reader may see why this threatens definability for a concurrent language with tensor: indeed a counter-strategy can then behave differently depending on the cause of an occurrence of the left $\mathbb{B}$.

We avoid the issue here, and only build $\multimap$ for $\mathbf{aPCF}_{\mathtt{por}}$ types. Interestingly the issue vanishes in Section 3: **Disj** supports disjunctive causality in both arenas and strategies.

Prestrategies on arenas. Strategies are certain event structures labeled by arenas. More formally, the labeling function is required to be a *map of event structures*:

▶ **Definition 4.** A **prestrategy** on arena A is a function on events $\sigma : S \to A$ *s.t.* σ is a **map of event structures**: it preserves configurations ($\forall x \in \mathscr{C}(S), \sigma\, x \in \mathscr{C}(A)$) and is locally injective ($\forall s_1, s_2 \in x \in \mathscr{C}(S), \sigma\, s_1 = \sigma\, s_2 \implies s_1 = s_2$).

Event structures and their maps form a category $\mathcal{E}$. Figure 1 displays such prestrategies $\sigma : S \to A$ – the event structure drawn is S, and events are annotated by their image in A through σ. *Strategies*, introduced in the next section, will be subject to further conditions.

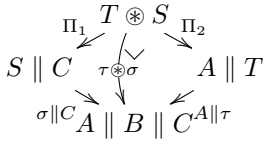
2.3 An interpretation of $\mathbf{aPCF}_{\mathtt{por}}$ as strategies

Following the methodology of denotational semantics, giving an interpretation of $\mathbf{aPCF}_{\mathtt{por}}$ consists in constructing a category out of certain prestrategies, with enough structure to interpret $\mathbf{aPCF}_{\mathtt{por}}$. For lack of space we can only sketch part of the construction, a more detailed account of the construction, originally from [17], can be found in [7].

A **prestrategy from** A **to** B is a prestrategy $\sigma : S \to A^\perp \parallel B$. Given also $\tau : T \to B^\perp \parallel C$, we wish to *compose* them. As usual in game semantics, this involves (1) parallel interaction where the strategies freely communicate, and (2) hiding. We focus on (1) first.

Interaction. The interaction of $\sigma : S \to A^\perp \parallel B$ and $\tau : T \to B^\perp \parallel C$ is an event structure $T \circledast S$, labeled by $A \parallel B \parallel C$ via $\tau \circledast \sigma : T \circledast S \to A \parallel B \parallel C$. The omitted definition [7] of $T \circledast S$ follows the lines of the third author's event structure semantics for parallel composition

in CCS [19]. It is uniquely determined (up to iso) as a *pullback* [7] in $\mathcal{E}$:



It is the "smallest" labeled event structure accommodating the constraints of σ and τ.

Hiding. Composition should yield a prestrategy from A to C. Accordingly events of $T \circledast S$ that map to A or C are available to the outside world, and called *visible*. On the other hand, those mapping to B are private synchronization events, dubbed *invisible*; that we wish to *hide*. The proposition below formalizes that event structures are expressive enough for that.

▶ **Proposition 5.** *Event structures* ***support hiding****: for E an event structure and V any subset of events, then there exists an event structure $E \downarrow V$ having V as events, and as configurations exactly those $x \cap V$ for $x \in \mathscr{C}(E)$.*

The components of $E \downarrow V$ (causality, conflict) are simply inherited from E.

For σ and τ as above, we first form $T \odot S$ as $T \circledast S \downarrow V$ with V the visible events. The **composition** $\tau \odot \sigma : T \odot S \to A^\perp \parallel C$ is simply the restriction of $\tau \circledast \sigma$; and is a prestrategy.

A category. Composition is associative up to isomorphism of prestrategies, where $\sigma_1 : S_1 \to A$ and $\sigma_2 : S_2 \to A$ are isomorphic (written $\sigma_1 \cong \sigma_2$) if there is an iso between S_1 and S_2 in $\mathcal{E}$ making the obvious triangle commute. Finally, for any arena A there is a **copycat prestrategy** $\text{ⓒ}_A : \mathbb{ⓒ}_A \to A^\perp \parallel A$, which has events and immediate conflict those of $A^\perp \parallel A$, and with causality that of $A^\perp \parallel A$ where additionally each positive event is set to depend on its negative counterpart on the other side. Details can be found in [17, 7].

Copycat is idempotent, but is not an identity in general. Rideau and Winskel [17] characterise the prestrategies for which it acts as an identity as the *strategies*:

▶ **Theorem 6.** *For $\sigma : S \to A^\perp \parallel B$, $\text{ⓒ}_B \odot \sigma \odot \text{ⓒ}_A \cong \sigma$ iff σ is a* ***strategy****: it is* ***receptive*** *(for $x \in \mathscr{C}(S)$, if $\sigma\, x \xrightarrow{a^-} \subset$ there is a unique $x \xrightarrow{s^-} \subset$ s.t. $\sigma\, s = a$) and* ***courteous*** *(if $s_1 \rightarrowtriangle_S s_2$ and $\mathsf{pol}(s_1) = +$ or $\mathsf{pol}(s_2) = -$, then $\sigma\, s_1 \rightarrowtriangle_{A^\perp \parallel B} \sigma\, s_2$).*

We mention (see [7]) that **CG** is a *compact closed category*, *i.e.* fit to interpret the *linear λ-calculus*. As $\mathbf{aPCF}_{\mathtt{por}}$ is *affine*, we work in a derived category of *negative strategies*.

Negative arenas and strategies. Negative arenas were defined earlier. We also have:

▶ **Definition 7.** A strategy $\sigma : S \to A^\perp \parallel B$ is **negative** iff $\forall s \in \min(S)$, $\mathsf{pol}(s) = -$.

There is a subcategory $\mathbf{CG}_-$ of **CG** with *negative arenas* as objects, and *negative strategies* as morphisms. The negativity assumption on strategies ensures that 1 is terminal (the only negative strategy on $A^\perp \parallel 1$, for A negative, is the empty strategy e_A), which allows us to interpret weakening. The tensor product is defined as $A \otimes B = A \parallel B$ on arenas, and as the obvious relabeling $\sigma_1 \otimes \sigma_2 : S_1 \parallel S_2 \to (A_1 \parallel A_2)^\perp \parallel B_1 \parallel B_2$ on strategies, for $\sigma_1 : S_1 \to A_1^\perp \parallel B_1$ and $\sigma_2 : S_2 \to A_2^\perp \parallel B_2$. Besides, $\mathbf{CG}_-$ has **products**: for $\sigma : S \to A^\perp \parallel B$ and $\tau : A^\perp \parallel C$ their pairing $\langle \sigma, \tau \rangle : S \,\&\, T \to A^\perp \parallel (B \,\&\, C)$ is the obvious labeling, and we have projections $\varpi_A : \mathbb{ⓒ}_A \to (A \,\&\, B)^\perp \parallel A$ and $\varpi_B : \mathbb{ⓒ}_B \to (A \,\&\, B)^\perp \parallel B$. For well-opened C, the negative strategies on $A^\perp \parallel B^\perp \parallel C$ and $A^\perp \parallel B \multimap C$ are exactly the same. Because $A \multimap B$ is only defined for well-opened B, $\mathbf{CG}_-$ is not monoidal closed; but well-opened arenas form an *exponential ideal*:

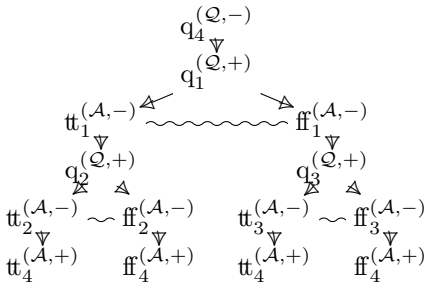


Figure 3 $\mathbf{if} : \mathbb{B}_1^\perp \parallel (\mathbb{B}_2 \,\&\, \mathbb{B}_3)^\perp \parallel \mathbb{B}_4$.

▶ **Proposition 8.** $\mathbf{CG}_{\_}$ *is a symmetric monoidal category with products, with* 1 *terminal. For* A *and well-opened* B, $A \multimap B$ *is an exponential object; and is well-opened.*

Following standard lines, we interpret $\mathbf{aPCF}_{\mathtt{por}}$ in $\mathbf{CG}_{\_}$: contexts $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ are interpreted as $[\![\Gamma]\!] = [\![A_1]\!] \parallel \cdots \parallel [\![A_n]\!]$. Types are interpreted as well-opened arenas, each type construction by its arena counterpart. Terms $\Gamma \vdash M : A$ are interpreted as negative strategies $[\![M]\!] \in \mathbf{CG}_{\_}([\![\Gamma]\!], [\![A]\!])$. Divergence $\bot$ is interpreted as the strategy (also written $\bot$) with no positive moves, closed by receptivity. Constants $\mathtt{tt}, \mathtt{ff}$ are the obvious strategies on $\mathbb{B}$. For $\Gamma \vdash M : \mathbb{B}$, $\Delta \vdash N_1, N_2 : \mathbb{B}$, $[\![\mathsf{if}\, M \,\mathsf{then}\, N_1 \mathsf{else}\, N_2]\!] = \mathbf{if} \odot ([\![M]\!] \otimes \langle [\![N_1]\!], [\![N_2]\!] \rangle)$, where $\mathbf{if}$ is given in Figure 3. Finally, parallel-or is the strategy on the right of Figure 1.

The interpretation $[\![-]\!]$ is adequate: for all $\vdash M : \mathbb{B}$, $M \Downarrow$ iff $[\![M]\!] \neq \bot$. However, we will be better equipped to prove that in Section 4. For now, the desired induction invariant for soundness $M \Downarrow v \Rightarrow [\![M]\!] = [\![v]\!]$ fails, as the model keeps track of too much intensional information: $[\![\mathsf{por}\, \mathtt{tt}\, \mathtt{tt}]\!]$ has two conflicting $\mathtt{tt}^+$ events, and is therefore not isomorphic to $[\![\mathtt{tt}]\!]$.

2.4 Disjunctive causality and observational determinism

We have seen that $\mathbf{CG}_{\_}$ is an adequate model of $\mathbf{aPCF}_{\mathtt{por}}$. However, it is not *intensionally fully abstract*: strategies distinguish more than terms of $\mathbf{aPCF}_{\mathtt{por}}$. In fact, as in [6] one can interpret *e.g.* linearly used boolean references in $\mathbf{CG}_{\_}$; those can obviously distinguish more than the input-output behaviour of terms. We can remove these by requiring conditions of visibility, well-bracketing and innocence as in [8]; and we will do so in Section 4.2.

One condition of [8] is however inadequate: *determinism*. Indeed, the strategy of Figure 1 is non-deterministic, and no deterministic strategy can implement parallel-or (indeed, all deterministic strategies on first-order types implement sequential functions [8]). Formalizing a notion of *observational determinism*, accepting the strategy for parallel-or but rejecting genuinely non-deterministic strategies, proved very challenging.

Disjunctive causality. As we pointed out in the introduction, the non-determinism of parallel-or comes from the incapacity of our version of event structures, sometimes referred to as *prime* event structures for disambiguation, to express *disjunctive causality*. So the reader may wonder: why do we bother with prime event structures at all? After all, there are alternatives. *General event structures* [18] allow events to be enabled in several distinct ways. We give their equivalent presentation [20] in terms of *configuration families*:

▶ **Definition 9.** A **configuration family (cf)** is a pair $(|\mathcal{A}|, \mathcal{A})$ (often just written $\mathcal{A}$) where $|\mathcal{A}|$ is a set of events, and $\mathcal{A}$ is a set of **configurations**, which are finite subsets of $|\mathcal{A}|$, such that $\emptyset \in \mathcal{A}$, and satisfying the two further axioms:

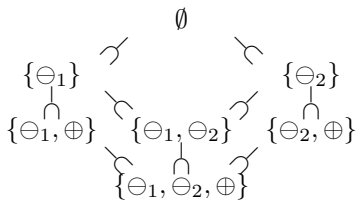


Figure 4 Disj. caus. in configuration families.

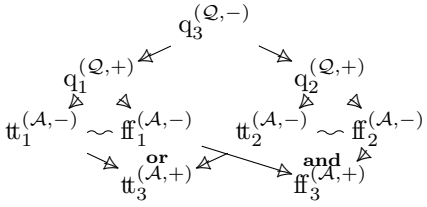


Figure 5 por as a disj. strategy.

- *Completeness.* For $x, y \in \mathcal{A}$, if $x \uparrow y$ (they are **compatible**, *i.e.* there exists $z \in \mathcal{A}$ such that $x \subseteq z$ and $y \subseteq z$), then $x \cup y \in \mathcal{A}$.
- *Coincidence-freeness.* If $x \in \mathcal{A}$, for all distinct $e_1, e_2 \in x$ there exists $y \subseteq x$ such that $y \in \mathcal{A}$ and $e_1 \in y \Leftrightarrow e_2 \notin y$.

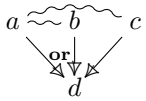
For every (prime) event structure A, $\mathscr{C}(A)$ is a configuration family. However configuration families are more general. In a configuration family, the same event can occur for different reasons. For instance, on the set of events $\{\ominus_1, \ominus_2, \oplus\}$, Figure 4 represents a configuration family where the event $\oplus$ can occur either because of $\ominus_1$ or $\ominus_2$, represented symbolically as in the diagram on the below – importantly, an occurrence of $\oplus$ carries no data on its effective cause.

With the same symbolic representation, the causally disjunctive nature of parallel-or is made explicit in Figure 5. It seems clear that this is a more accurate description of parallel-or. In particular, it is *deterministic*, in a sense to be made formal in the next section.

Are configuration families sufficiently expressive as a basis to construct a category of strategies, as we did with prime event structures in the previous section? Recall that for that we used two properties of event structures: that the corresponding category $\mathcal{E}$ has *pullbacks* (for interactions), and then that event structures support *hiding*. We will show very soon that the first criterion holds. However, we run into an issue for the second:

▶ **Proposition 10.** *Configuration families do* not *support hiding. There is a cf $\mathcal{A}$ and $V \subseteq |A|$ such that the set $\{x \cap V \mid x \in \mathcal{A}\}$ is not a configuration family.*

Proof. Set $|\mathcal{A}| = \{a, b, c, d\}$, and configurations those specified by: (d is caused either by a, or by b and c together). With $V = \{b, c, d\}$, the obtained hidden set fails completeness: it contains $\{b\}, \{d\}, \{b, c, d\}$ but not $\{b, d\}$.



◀

It is part for this reason that prime event structures are used when building categories of games and strategies – as indeed, hiding is crucial. Removing either coincidence-freeness or completeness loses the correspondence with general event structures [20], and leads to various pathologies further down the road – for instance we lose the key lemma:

▶ **Lemma 11.** *Let $\mathcal{A}$ a set of configurations on $|\mathcal{A}|$ satisfying completeness. Then, it is coincidence-free iff for any $x, y \in \mathcal{A}$* s.t. *$x \subseteq y$, there is a covering chain in $\mathcal{A}$: $x \mathrel{-\!\!\subset} \ldots \mathrel{-\!\!\subset} y$.*

3 Disjunctive deterministic games

In this section we introduce the main contribution of the paper, a category **Disj** of disjunctive deterministic strategies supporting the interpretation of parallel-or. This relies on a notion of *deterministic* configuration families, and the observation that those *do* support hiding.

3.1 Deterministic configuration families

First we adjoin cfs with *polarities*, and define determinism (for Player).

▶ **Definition 12.** A cf **with polarities (cfp)** (*resp.* **partial polarities (cfpp)**) is a cf $\mathcal{A}$ with $\mathsf{pol}_{\mathcal{A}} : |A| \to \{-,+\}$ (*resp.* $\mathsf{pol}_{\mathcal{A}} : |A| \to \{-,0,+\}$).

A cfpp $\mathcal{A}$ is **deterministic** (dcfp/dcfpp) iff for all $x \subseteq^p_{\mathcal{A}} y$ and $x \subseteq z$, we have $y \cup z \in \mathcal{A}$ – we write $x \subseteq^p_{\mathcal{A}} y$ (resp. $x \subseteq^-_{\mathcal{A}} y, x \subseteq^+_{\mathcal{A}} y$) to mean that $x \subseteq y$ and $\mathsf{pol}(y \setminus x) \subseteq \{0,+\}$ (resp. $\mathsf{pol}_{\mathcal{A}}(y \setminus x) \subseteq \{-\}$, $\mathsf{pol}_{\mathcal{A}}(y \setminus x) \subseteq \{+\}$).

This means that only Opponent makes irreversible choices regarding the evolution of the play. We show that dcfpps support hiding of neutral events. In the sequel, given a dcfpp $\mathcal{A}$, V always denotes the set of non-neutral events in $|A|$. We write $\mathcal{A}_\downarrow$ for the candidate cf on events V with configurations those of the form $x_\downarrow = x \cap V$ for $x \in \mathcal{A}$. The following key lemma is proved by successive applications of determinism and completeness.

▶ **Lemma 13.** *For $x \in \mathcal{A}_\downarrow$, $y \in \mathcal{A}$ such that $x \subseteq y_\downarrow$, for any $x' \in \mathcal{A}$ a witness for x (*i.e. *$x'_\downarrow = x$), there is $y \subseteq y' \in \mathcal{A}$ such that $x' \subseteq y'$.*

From Lemma 13 follows that for $x, y \in \mathcal{A}$, if $x_\downarrow \uparrow y_\downarrow$ in $\mathcal{A}_\downarrow$, then $x \uparrow y$ in $\mathcal{A}$. Using that observation together with Lemma 13 and determinism, it is straightforward to prove:

▶ **Proposition 14.** *For $\mathcal{A}$ a dcfpp, $\mathcal{A}_\downarrow$ is a dcfp.*

3.2 Deterministic disjunctive strategies and composition

A **pregame** is simply a cfp – for $\mathcal{A}, \mathcal{B}$ cfps, $\mathcal{A}^\perp$ has the same events and configurations as $\mathcal{A}$ but inverted polarity. We will also write $\mathcal{B}^0$ for the cfpp with the same configurations as $\mathcal{B}$ but polarity set as globally 0. The cf $\mathcal{A} \parallel \mathcal{B}$ has events the tagged disjoint union, and configurations $x_A \parallel x_B$ for $x_A \in \mathcal{A}$, $x_B \in \mathcal{B}$; $\mathcal{A} \,\&\, \mathcal{B}$ has the same events as $\mathcal{A} \parallel \mathcal{B}$, but only those configurations with one side empty.

Unlike in Definition 4, *disjunctive prestrategies* are simply substructures of the (pre)games.

▶ **Definition 15.** A **prestrategy** on $\mathcal{A}$ is a dcfp σ on $|\mathcal{A}|$, such that $\sigma \subseteq \mathcal{A}$ – written $\sigma : \mathcal{A}$.

▶ **Example 16.** On events (with polarities) $\{\mathrm{q}^-, \mathrm{tt}^+, \mathrm{ff}^+\}$, $\mathscr{C}(\mathbb{B})$ is a cfp – that by abuse of notation we still write $\mathbb{B}$. Then, Figure 5 denotes a prestrategy on $\mathbb{B}^\perp \parallel \mathbb{B}^\perp \parallel \mathbb{B}$.

Though disjunctive prestrategies are not primarily defined as maps, it will be helpful in composing them that they *can* be. Given two cfs $\mathcal{A}$ and $\mathcal{B}$, a **map from $\mathcal{A}$ to $\mathcal{B}$** is a function on events $f : |\mathcal{A}| \to |\mathcal{B}|$ which preserves configurations and is locally injective. Configuration families and their maps forms a category **Fam**. Note that if A is an event structure, $\mathscr{C}(A)$ is a configuration family on $|A|$. The definition of maps of cfs is compatible with that of maps of event structures, making $\mathscr{C}(-) : \mathcal{E} \to \mathbf{Fam}$ a full and faithful functor. Finally, a prestrategy $\sigma : \mathcal{A}$ on $\mathcal{A}$ can be regarded as an identity-on-events **Fam**-morphism $\sigma \to \mathcal{A}$.

Like $\mathcal{E}$, **Fam** has pullbacks. The **interaction** of σ on $\mathcal{A}^\perp \parallel \mathcal{B}$ and τ on $\mathcal{B}^\perp \parallel \mathcal{C}$ is the pullback of identity-on-events maps $\sigma \parallel \mathcal{C} \to \mathcal{A} \parallel \mathcal{B} \parallel \mathcal{C}$ and $\mathcal{A} \parallel \tau \to \mathcal{A} \parallel \mathcal{B} \parallel \mathcal{C}$.

▶ **Proposition 17.** *The pullback above is (up to iso) the cf $\tau \circledast \sigma$ with events $|\mathcal{A}| \parallel |\mathcal{B}| \parallel |\mathcal{C}|$ and configurations those $x \in (\sigma \parallel \mathcal{C}) \cap (\mathcal{A} \parallel \tau)$ that are **secured**: for distinct $p_1, p_2 \in x$ there is $x \supseteq y \in (\sigma \parallel \mathcal{C}) \cap (\mathcal{A} \parallel \tau)$* s.t. *$p_1 \in y \Leftrightarrow p_2 \notin y$.*

Equivalently, $x \in (\sigma \parallel \mathcal{C}) \cap (\mathcal{A} \parallel \tau)$ is secured iff it has a *covering chain*, *i.e.* a sequence: $\emptyset = x_0 \lessdot x_1 \lessdot \ldots \lessdot x_n = x$ where for all $0 \leq i \leq n$, $x_i \in (\sigma \parallel \mathcal{C}) \cap (\mathcal{A} \parallel \tau)$.

We regard $\tau \circledast \sigma$ as a cfpp by setting as polarities those of $\mathcal{A}^\perp \parallel \mathcal{B}^0 \parallel \mathcal{C}$. To use Proposition 14 and finish defining composition, $\tau \circledast \sigma$ needs to be deterministic; a sufficient condition for that is that σ and τ are *receptive*. A prestrategy σ on $\mathcal{A}$ is **receptive** iff for all $x \in \sigma$, if $x \cup \{a^-\} \in \mathcal{A}$ then $x \cup \{a^-\} \in \sigma$.

▶ **Proposition 18.** *If σ and τ are receptive prestrategies, then $\tau \circledast \sigma$ is a* deterministic *cfpp. Then, $\tau \odot \sigma = (\tau \circledast \sigma)_\downarrow$ is a receptive prestrategy on $\mathcal{A}^\perp \parallel \mathcal{C}$.*

Hence, composition is well-defined on receptive prestrategies – it is also associative. To get a category, we now prove the copycat strategy to be an identity.

3.3 Copycat and the compact closed category Disj

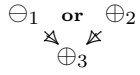
We cannot replicate in pregames the definition of copycat sketched in Section 2.3. However, as observed in [22, 7], if A is an arena, the *configurations* of $\mathbb{CC}_A$ are those configurations of $A^\perp \parallel A$, necessarily of the form $x_l \parallel x_r$, such that every positive event of x_r (*w.r.t.* A) is already in x_l, and every positive event in x_l (*w.r.t.* $A^\perp$) is already in x_r. In other words, $x_l \supseteq_A^- x_l \cap x_r \subseteq_A^+ x_r$, written $x_r \sqsubseteq_A x_l$ in [22, 7] and and referred to as the "Scott order".

Accordingly, on a pregame $\mathcal{A}$, given $x, y \in \mathcal{A}$ we write $x \sqsubseteq_\mathcal{A} y$ iff $x \cap y \in \mathcal{A}$ and the relation above holds. The candidate prestrategy **copycat** $\mathfrak{c}_A$ comprises all $x \parallel y \in \mathcal{A}^\perp \parallel \mathcal{A}$ *s.t.* $y \sqsubseteq_\mathcal{A} x$. Indeed $\mathfrak{c}_\mathcal{A}$ is a configuration family and is receptive; but prestrategies must be *deterministic*. It turns out that as in [21], $\mathfrak{c}_\mathcal{A}$ is only deterministic when $\mathcal{A}$ is *race-free*:

▶ **Proposition 19.** *Let $\mathcal{A}$ be a pregame. Then, $\mathfrak{c}_\mathcal{A}$ is a prestrategy iff $\mathcal{A}$ is **race-free**: for all $x, y, z \in \mathcal{A}$ such that $x \subseteq_\mathcal{A}^+ y$ and $x \subseteq_\mathcal{A}^- z$, we have $y \uparrow_\mathcal{A} z$.*

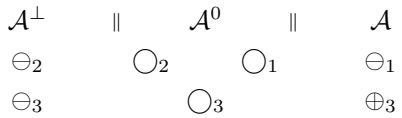
We aim to reproduce Theorem 6 in this new setting, and characterise the prestrategies left invariant under composition with copycat. However, there is a new subtlety here: for arbitrary $\mathcal{A}$, $\mathfrak{c}_\mathcal{A}$ might not even be idempotent!

▶ **Example 20.** Consider the cfp on events $A = \{\ominus_1, \oplus_2, \oplus_3\}$ given on the below.



This is a race-free pregame, so by Proposition 19, $\mathfrak{c}_\mathcal{A} \odot \mathfrak{c}_\mathcal{A}$ is a prestrategy on $\mathcal{A}^\perp \parallel \mathcal{A}$. However, it is *distinct from* $\mathfrak{c}_\mathcal{A}$.

Indeed, the following configuration of $\mathcal{A}^\perp \parallel \mathcal{A}^0 \parallel \mathcal{A}$ belongs to $\mathfrak{c}_\mathcal{A} \circledast \mathfrak{c}_\mathcal{A}$:



After hiding, there is a change in the causal history of $\oplus_3$, not authorized by $\mathfrak{c}_\mathcal{A}$ but authorised by $\mathcal{A}$: it is caused by $\ominus_2$ on the left, but $\ominus_1$ on the right. This issue comes from the fact that in $\mathcal{A}$, the same event can be caused either by a positive or a negative move.

Such behaviour will be banned in games:

▶ **Proposition 21.** *If $\mathcal{A}$ is a race-free pregame, the following are equivalent: (1) $\mathfrak{a}_{\mathcal{A}}$ is idempotent, (2) $\sqsubseteq_{\mathcal{A}}$ is a partial order, (3) $\mathcal{A}$ is **co-race-free**: for all $x, y, z \in \mathcal{A}$ with $x \supseteq^{-} y, x \supseteq^{+} z$, we have $y \cap z \in \mathcal{A}$.*

A **game** will be a race-free, co-race-free pregame. Copycat on any game is an idempotent prestrategy – *strategies* are those prestrategies that compose well with copycat. We prove:

▶ **Theorem 22.** *Let σ be a receptive prestrategy on $\mathcal{A}^{\perp} \parallel \mathcal{B}$. Then, $\mathfrak{a}_{\mathcal{B}} \odot \sigma \odot \mathfrak{a}_{\mathcal{A}} = \sigma$ iff σ is **courteous**: for any $x \multimap\subset x \cup \{a_1^+\} \multimap\subset x \cup \{a_1^+, a_2^+\}$ in σ, if $x \cup \{a_2^+\} \in \mathcal{A}^{\perp} \parallel \mathcal{B}$, then $x \cup \{a_2^+\} \in \sigma$ as well. In this case, σ is a **strategy**, written: $\sigma : \mathcal{A}^{\perp} \parallel \mathcal{B}$.*

It follows that there is a category **Disj** with games as objects, and strategies $\sigma : \mathcal{A}^{\perp} \parallel \mathcal{B}$ as morphisms from $\mathcal{A}$ to $\mathcal{B}$. It is fairly easy to show that just as **CG**, this category is compact closed. But unlike **CG**, **Disj** supports a deterministic interpretation of parallel-or: indeed the set of configurations of $\mathbb{B}^{\perp} \parallel \mathbb{B}^{\perp} \parallel \mathbb{B}$ denoted by the diagram of Figure 5 is a strategy.

3.4 An SMCC and deterministic interpretation of $\mathrm{aPCF}_{\mathtt{por}}$

As for **CG**, **Disj** lacks structure to interpret $\mathbf{aPCF}_{\mathtt{por}}$: the family of bottom strategies $e_{\mathcal{A}} : \mathcal{A}^{\perp} \parallel 1$ (where again 1 is the empty game) fails naturality. As before, we hence restrict **Disj** to a subcategory of *negative* games and strategies.

▶ **Definition 23.** A cfp $\mathcal{A}$ is **negative** if any non-empty $x \in \mathcal{A}$ includes a negative event.

Copycat on negative $\mathcal{A}$ is negative and negative strategies are stable under composition; so there is a subcategory $\mathbf{Disj}_{-}$ of **Disj** with negative games as objects and negative strategies as morphisms, inheriting a symmetric monoidal structure from **Disj**. Moreover 1 is terminal, and $\mathbf{Disj}_{-}$ has products given by $\mathcal{A} \,\&\, \mathcal{B}$.

To prove the monoidal closure, as in **CG**, we have to deal with the fact that for $\mathcal{A}$ and $\mathcal{B}$ negative, $\mathcal{A}^{\perp} \parallel \mathcal{B}$ is not necessarily negative, so we define:

▶ **Definition 24.** Let $\mathcal{A}$ and $\mathcal{B}$ be two negative games. The game $\mathcal{A} \multimap \mathcal{B}$ has the same events (and polarity) as $\mathcal{A}^{\perp} \parallel \mathcal{B}$, but non-empty configurations those $x_A \parallel x_B \in \mathcal{A}^{\perp} \parallel \mathcal{B}$ such that x_B is non-empty (and hence includes a negative event).

Recall from Definition 3 that the arrow arena $A \multimap B$ was only defined for B *well-opened* (*i.e.* with at most one minimal event). This was to avoid constructing arenas for types like $\mathbb{B} \multimap (\mathbb{B} \otimes \mathbb{B})$ (invalid for $\mathbf{aPCF}_{\mathtt{por}}$, but valid in an extension with a tensor type), where the left hand side $\mathbb{B}$ can be opened because of either of the right hand side occurrences of $\mathbb{B}$. In **Disj**, when constructing $\mathcal{A} \multimap \mathcal{B}$ there is no well-openness condition on $\mathcal{B}$: exploiting disjunctive causality we can express that $\mathcal{A}$ is opened only after *some* event of $\mathcal{B}$ – it does not matter which one. The negative strategies in $\mathcal{A}^{\perp} \parallel \mathcal{B}^{\perp} \parallel \mathcal{C}$ and $\mathcal{A}^{\perp} \parallel \mathcal{B} \multimap \mathcal{C}$ are the same, from which (using the compact closed structure of **Disj**) it follows that $\mathbf{Disj}_{-}$ is monoidal closed, instead of only having an exponential ideal. Like $\mathbf{CG}_{-}$, it supports an adequate interpretation of $\mathbf{aPCF}_{\mathtt{por}}$. Unlike $\mathbf{CG}_{-}$, its strategies are *deterministic*.

4 Glueing CG_{-} and Disj_{-}, and full abstraction

By moving from $\mathbf{CG}_{-}$ to $\mathbf{Disj}_{-}$, we gain determinism. However, $\mathbf{Disj}_{-}$ is *not* intensionally fully abstract – there are some strategies that distinguish syntactically undistinguishable terms

of $\mathbf{aPCF}_{\mathtt{por}}$. For instance, one has a strategy on $(\mathbb{B} \multimap \mathbb{B} \multimap \mathbb{B}) \multimap \mathbb{B}$ that calls its argument, feeds it $\mathtt{tt}$ as first argument, and $\mathtt{tt}$ as second argument only if the first argument has been evaluated; it then copies the final result to toplevel. Doing so, it distinguishes the observationally equivalent $\lambda xy.\,\mathrm{if}\ x\ \mathrm{then}\ (\mathrm{if}\ y\ \mathrm{then}\ \mathtt{tt}\ \mathrm{else}\ \bot)\ \mathrm{else}\ \bot$ and $\lambda xy.\,\mathrm{if}\ y\ \mathrm{then}\ (\mathrm{if}\ x\ \mathrm{then}\ \mathtt{tt}\ \mathrm{else}\ \bot)\ \mathrm{else}\ \bot$.

Getting rid of such strategies is the responsibility of the concept of *innocence* in Hyland-Ong games [13]. But the *P-views* of innocent strategies carry precisely the *causal* information that we have lost when moving from $\mathbf{CG}_{\_}$ to $\mathbf{Disj}_{\_}$! This causal information was crucial in [8] to give concurrent versions of well-bracketing and innocence, so as to capture the behaviour of parallel **PCF** strategies. So, the deterministic account of $\mathbf{aPCF}_{\mathtt{por}}$ is not enough; the causal (where innocence and well-bracketing are defined) and deterministic models should be related, to establish in what sense the causal model is already "observably" deterministic.

4.1 A glued games model

For any arena A (Definition 2), $\mathscr{C}(A)$ is a game – it is race-free and co-race-free. Moreover, arena constructions and game constructions match: for any A, B, $\mathscr{C}(A^{\perp}) = \mathscr{C}(A)^{\perp}$, $\mathscr{C}(A \parallel B) = \mathscr{C}(A) \parallel \mathscr{C}(B)$; for negative A, B we have $\mathscr{C}(A \,\&\, B) = \mathscr{C}(A) \,\&\, \mathscr{C}(B)$, and if B is well-opened, $\mathscr{C}(A \multimap B) = \mathscr{C}(A) \multimap \mathscr{C}(B)$. Therefore, we will just take the objects of our glued games model to be arenas. The strategies, though, will be strategies in *both categories*.

▶ **Definition 25.** Let A be an arena. A strategy $\sigma : S \to A$ is **observationally deterministic (odet)** if (1) the **display** of σ, $\mathcal{O}(\sigma) = \{\sigma\, x \mid x \in \mathscr{C}(S)\}$ is a disjunctive deterministic strategy on $\mathscr{C}(A)$ (in **Disj**). It follows then that σ is also a **Fam**-morphism $\sigma : \mathscr{C}(S) \to \mathcal{O}(\sigma)$. We also ask that it has (2) the **configuration extension property**: for any $x \in \mathscr{C}(S)$, if $\sigma\, x \subseteq y \in \mathcal{O}(\sigma)$, then there is $x \subseteq x' \in \mathscr{C}(S)$ such that $\sigma\, x' = y$.

The configuration extension property ensures that two causal realizations of the same displayed configuration have bisimilar futures. The display operation yields a coarser equivalence on strategies: odet strategies $\sigma, \tau : A$, are **display-equivalent**, written $\sigma \approx \tau$, if $\mathcal{O}(\sigma) = \mathcal{O}(\tau)$. This coarser equivalence is key to ensure soundness of our interpretation, as the interpretation in $\mathbf{CG}_{\_}$ of $\mathsf{por}\ \mathtt{tt}\ \mathtt{tt}$ and $\mathtt{tt}$ are *not* isomorphic, but only display-equivalent.

▶ **Theorem 26.** *There is a compact closed category* **Odet** *with arenas as objects and odet strategies up to* $\approx$ *as morphisms; with a symmetric monoidal subcategory* $\mathbf{Odet}_{\_}$ *of negative arenas and negative strategies with products and* 1 *terminal. It has well-opened arenas as an exponential ideal. Finally,* $\mathbf{Odet}_{\_}$ *supports an adequate interpretation of* $\mathbf{aPCF}_{por}$.

Proof. For odet σ and τ, $\mathcal{O}(\tau \odot \sigma) = \mathcal{O}(\tau) \odot \mathcal{O}(\sigma)$: $\subseteq$ is direct, $\supseteq$ exploits the extension property for σ and τ. So, $\mathcal{O}(\tau \odot \sigma)$ is a deterministic disjunctive strategy, and it has the extension property – hence $\tau \odot \sigma$ is odet. In general, $\mathcal{O}$ links the structure of **CG** to that of **Disj**, forming **Odet**. We sketch adequacy.

Soundness. For all $M \Downarrow v$, $[\![M]\!] = [\![v]\!]$ (by induction on the evaluation derivations).

Adequacy. We prove by induction on the size of M that if $M \Uparrow$, then $[\![M]\!]$ is bottom. The size is an adequate measure because as $\mathbf{aPCF}_{\mathtt{por}}$ is affine, substitution is non-copying and each induction step is on a strictly smaller term. ◀

As a result, interpretations in $\mathbf{CG}_{\_}$ and **Disj** are also adequate. We have accommodated the causal representation of programs permitted by **CG** and the determinism of **Disj**. Now, it remains to import the causal conditions on strategies of [8], and prove full abstraction.

4.2 Conditions

We now recall *innocence* and *well-bracketing*, introduced in [8]. Formulated for *deterministic strategies* (in the sense of $\mathbf{CG}_{-}$), those would not suffice to prove full abstraction for a genuinely non-deterministic language. For that, further conditions are needed to ensure the locality of conflicts – those will appear in the first author's forthcoming PhD thesis. However, we will show in Lemma 31 that for $\mathbf{aPCF}_{\mathtt{por}}$, distinguishable strategies can be distinguished via *deterministic* (in the sense of $\mathbf{CG}_{-}$) contexts, so these simpler conditions will suffice.

All our conditions rely crucially on the notion of *grounded causal chain*.

▶ **Definition 27.** A **grounded causal chain** (**gcc**) in S is a sequence $\rho = \rho_0 \rightarrowtriangle \ldots \rightarrowtriangle \rho_n$ where ρ_0 is minimal. The set of gccs of S is written $\mathrm{gcc}(S)$.

By courtesy, gccs of strategies on arenas are always alternating. They give a notion of *thread* of a concurrent strategy. Two gccs $\rho, \rho' \in \mathrm{gcc}(S)$ are **forking** when $\rho \cup \rho'$ is consistent in S, and there is $k \in \mathbb{N}$ s.t. $\rho_i = \rho'_i$ for $i < k$ and $\{\rho_i\}_{i \geq k}$ and $\{\rho'_j\}_{j \geq k}$ are non-empty and disjoint. They are **negatively forking** when $\mathsf{pol}(\rho_{k+1}) = \mathsf{pol}(\rho'_{k+1}) = -$, **positively forking** otherwise. Two forking gccs ρ, ρ' are **joined** at $s \in S$ when $\rho_\omega \rightarrowtriangle s$ and $\rho'_\omega \rightarrowtriangle s$. ($\rho_\omega$ refers to the last event of ρ) as shown in this picture:

$$\rho_0 \rightarrowtriangle \ldots \rightarrowtriangle \rho_k \begin{array}{l} \nearrow \rho_{k+1} \rightarrowtriangle \ldots \rightarrowtriangle \rho_\omega \searrow \\ \searrow \rho'_{k+1} \rightarrowtriangle \ldots \rightarrowtriangle \rho'_\omega \nearrow \end{array} s$$

Innocence. *Innocence* enforces independence between threads forked by Opponent.

▶ **Definition 28.** Let $\sigma : S \to A$ be a strategy on a arena A. It is **innocent** when it is:
Visible: The image $\sigma\,\rho$ of a gcc (regarding $\rho \in \mathrm{gcc}(S)$ as a set) is a configuration of A.
Preinnocent: Two negatively forking gccs are never joined.

Preinnocence is a *locality* condition, forcing Player to deal independently with threads forked by Opponent – in the sequential case, it coincides with Hyland-Ong innocence. In the concurrent case though, it is not by itself stable under composition; that is where visibility comes in. Together they are preserved under composition and under all the operations on strategies involved in the interpretation of $\mathbf{aPCF}_{\mathtt{por}}$ in $\mathbf{CG}_{-}$ (and hence $\mathbf{Odet}_{-}$).

Well-bracketing. Traditionally, *well-bracketing* in game semantics rules out strategies that behave like a control operator, manipulating the call stack. It exploits the question/answer labelling, reminiscent of the function calls/returns. In arenas from $\mathbf{aPCF}_{\mathtt{por}}$ types, answers to the same question are always conflicting, so every consistent set has at most one answer to any question. A consistent set X is **complete** when it has exactly one answer to any question. A question is **pending** in a set X if it has no answer in X and maximally so in X.

▶ **Definition 29.** A strategy $\sigma : S \to A$ is **well-bracketed** when (1) if $a \in S$ answers $q \in S$, then q is pending in $[a]$ and (2) for $\rho, \rho' \in \mathrm{gcc}(S)$ forking at $\rho_k = \rho'_k$, and joined, the segments $\rho_{>k}$ and $\rho_{>k'}$ must be complete.

The affinity condition of [8] comes for free here, thanks to the conflict in $\mathbb{B}$. Well-bracketing is proved stable under composition and other operations in [8]. A $\mathbf{aPCF}_{\mathtt{por}}$**-strategy** is a negative, innocent, well-bracketed and odet strategy.

▶ **Theorem 30.** *There is a symmetric monoidal category* **PorStrat** *of negative arenas and* $\mathbf{aPCF}_{por}$*-strategies, with products, a terminal object and an exponential ideal comprising* $\mathbb{B}$*. Moreover, the interpretation of* $\mathbf{aPCF}_{por}$ *in* $\mathbf{Odet}_{-}$ *factors through* $\mathbf{PorStrat} \subseteq \mathbf{Odet}_{-}$*.*

From now on, all strategies will be assumed to be $\mathbf{aPCF}_{\mathtt{por}}$-strategies.

4.3 Intensional full abstraction

Two strategies $\sigma, \tau : A$ are **observationally equivalent** ($\sigma \simeq_{\mathrm{obs}} \tau$) when for all strategies $\alpha : A^{\perp} \parr \mathbb{B}$, $\alpha \odot \sigma \approx \alpha \odot \tau$. In fact, there is no need to quantify over *all* strategies. A **path-strategy** is a strategy $\sigma : S \to A$ such that there exists a configuration $x \in \mathscr{C}(S)$ that contains all the positive moves of S. Any distinguishing strategy yields by restriction a distinguishing path-strategy – this would fail without affinity, as the restriction would fail uniformity [8], and enforcing uniformity would make it non-deterministic.

▶ **Lemma 31.** *Two distinguishable strategies can be distinguished by a path-strategy.*

▶ **Lemma 32.** *Every path-strategy can be defined by a term of* **aPCF** *up to* $\simeq_{\mathrm{obs}}$.

Proof. As in [8] since path-strategies are deterministic – the only difference is the necessity to distribute the context among the subterms to ensure affine typing. ◀

A corollary is that in an affine setting, por adds no distinguishing power: two $\mathbf{aPCF}_{\mathtt{por}}$ terms are observationally equivalent if and only if they cannot be distinguished by a context from affine **PCF** without por. Intensional full abstraction follows by standard techniques:

▶ **Theorem 33.** *The interpretation of* $\mathbf{aPCF}_{por}$ *into* **PorStrat** *is intensionally fully abstract: it preserves and reflects observational equivalence (see Section 2.1).*

5 Conclusion

This leaves many avenues for further investigation. On the semantic front, we plan among other applications to exploit **Odet** to model non-interfering concurrent languages. On the foundational front we need to understand better how the present approach relates to the treatment of disjunctive causality in edcs [9].

Acknowledgments. We are grateful to Tamás Kispéter for interesting discussions about strategies as general event structures.

References

1 Samson Abramsky, Kohei Honda, and Guy McCusker. A fully abstract game semantics for general references. In *LICS, Indianapolis, Indiana, USA, June 21-24, 1998*, pages 334–344. IEEE Computer Society, 1998.
2 Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000.
3 Samson Abramsky and Guy McCusker. Linearity, sharing and state: a fully abstract game semantics for idealized algol with active expressions. *ENTCS*, 3:2–14, 1996.
4 Samson Abramsky and Paul-André Melliès. Concurrent games and full completeness. In *LICS, Trento, Italy, July 2-5, 1999*, pages 431–442. IEEE Computer Society, 1999.
5 Robert Cartwright, Pierre-Louis Curien, and Matthias Felleisen. Fully abstract semantics for observably sequential languages. *Inf. Comput.*, 111(2):297–401, 1994.
6 Simon Castellan and Pierre Clairambault. Causality vs. interleavings in concurrent game semantics. In *CONCUR 2016, August 23-26, 2016, Québec City, Canada*, pages 32:1–32:14, 2016.
7 Simon Castellan, Pierre Clairambault, Silvain Rideau, and Glynn Winskel. Games and strategies as event structures, 2016. Submitted, `https://arxiv.org/abs/1604.04390`.

8 Simon Castellan, Pierre Clairambault, and Glynn Winskel. The parallel intensionally fully abstract games model of PCF. In *LICS, Kyoto, Japan, July 6-10, 2015*, pages 232–243. IEEE Computer Society, 2015.

9 Marc de Visme and Glynn Winskel. Strategies with Parallel Causes. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 41:1–41:21, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2017.41.

10 Dan R. Ghica. Applications of game semantics: From program analysis to hardware synthesis. In *LICS, 11-14 August 2009, Los Angeles, CA, USA*, pages 17–26. IEEE Computer Society, 2009.

11 Dan R. Ghica and Andrzej S. Murawski. Angelic semantics of fine-grained concurrency. *Ann. Pure Appl. Logic*, 151(2-3):89–114, 2008.

12 Tom Hirschowitz. Full abstraction for fair testing in CCS (expanded version). *Logical Methods in Computer Science*, 10(4), 2014.

13 J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.

14 James Laird. Full abstraction for functional languages with control. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 – July 2, 1997*, pages 58–67. IEEE Computer Society, 1997.

15 Paul-André Melliès and Samuel Mimram. Asynchronous games: Innocence without alternation. In *CONCUR, Lisbon, Portugal, September 3-8, 2007*, volume 4703 of *Lecture Notes in Computer Science*, pages 395–411. Springer, 2007.

16 Gordon D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977.

17 Silvain Rideau and Glynn Winskel. Concurrent strategies. In *LICS, June 21-24, 2011, Toronto, Ontario, Canada*, pages 409–418, 2011.

18 Glynn Winskel. *Events in computation*. PhD Thesis, Edinburgh University, 1980.

19 Glynn Winskel. Event structure semantics for CCS and related languages. In *ICALP, Aarhus, Denmark, July 12-16, 1982, Proceedings*, pages 561–576, 1982.

20 Glynn Winskel. Event structures. In *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986*, pages 325–392, 1986.

21 Glynn Winskel. Deterministic concurrent strategies. *Formal Asp. Comput.*, 24(4-6):647–660, 2012.

22 Glynn Winskel. Strategies as profunctors. In *FOSSACS, Held as Part of ETAPS 2013, Rome, Italy, March 16-24, 2013*, pages 418–433, 2013.

There Is Only One Notion of Differentiation*

J. Robin B. Cockett[1] and Jean-Simon Lemay[2]

1 **Department of Computer Science, University of Calgary, Calgary, AB, Canada**
robin@ucalgary.ca

2 **Department of Mathematics and Statistics, University of Calgary, Calgary, AB, Canada**
jeansimon.lemay@ucalgary.ca

Abstract

Differential linear logic was introduced as a syntactic proof-theoretic approach to the analysis of differential calculus. Differential categories were subsequently introduce to provide a categorical model theory for differential linear logic. Differential categories used two different approaches for defining differentiation abstractly: a deriving transformation and a coderiliction. While it was thought that these notions could give rise to distinct notions of differentiation, we show here that these notions, in the presence of a monoidal coalgebra modality, are completely equivalent.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Differential Categories, Linear Logic, Coalgebra Modalities, and Bialgebra Modalities

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.13

1 Introduction

Differential linear logic [5, 6] was introduced as a syntactic proof-theoretic approach to the analysis of differential calculus. Differential categories [2] were subsequently introduced to provide a categorical model theory for differential linear logic. In [2] two approaches to defining differentiation abstractly were introduced, these being based on a deriving transformation and a coderiliction. Fiore in [7] proposed an alternative approach to differentiation using what he called a *creation operator* – this was an operator having the same type as a deriving transformation but with a rather different axiomatization. He argued that, in the presence of a monoidal coalgebra modality and biproducts, his notion of differentiation provided a substantially better theory. However, as will be shown below, the notion of a creation operator is actually completely equivalent to that of a deriving transformation (with the interchange rule) as presented in [2]. Furthermore, these are, in that setting, equivalent to having a codereliction.

To show these notions are all equivalent it is necessary to explore, in some detail, the relation between a codereliction – essentially Fiore's "creation *map*" – and a deriving transformation (essentially Fiore's "creation *operator*"). It is straightforward to show that having a codereliction always implies the presence of a deriving transformation. The reverse, however, is more difficult and this is the main technical accomplishment of this paper. Its consequence is that contrary to the suggestion put forward in [7] there is only one notion of differentiation in linear logic.

* This work was partially supported by NSERC, Canada.

First we should pause to consider Fiore's argument that one may as well work in a setting with biproducts. He argued that, as one can always complete an additive category to a category with biproducts, one may as well assume biproducts are present. Of course, when one considers differentiation, one also needs to be able to extend the coalgebra modality. It is standard that one can extend a *monoidal* coalgebra modality to the biproduct completion using the Seely isomorphisms. However, when the modality is *not* monoidal, there is apparently no reason why it should extend. Significantly, the deriving transformation assumed in [2] did not assume the coalgebra modality was monoidal. Thus, if one is to entertain these more basic modalities, one must be cautious about using Fiore's argument. Here we certainly take such modalities seriously.

The paper starts by reminding the reader of the "original" definition of a differential category. This relies on the idea of a coalgebra modality and a deriving transformation. Familiarity with linear logic may tempt one to think that this is the "exponential" modality of linear logic but it is not. It is a strictly weaker notion as the modality is not assumed monoidal. There are many familiar examples of differential categories based on a monoidal coalgebra modality [2] – such as (the opposite of) the free symmetric algebra monad on vector spaces. The reader may wonder, however, whether there are any significant examples of differential categories based a mere coalgebra modality. Two compelling examples – by no means the only ones – are given by smooth functions via the free $\mathcal{C}^\infty$-ring over vector spaces (as mentioned in [2]) and the free Rota-Baxter algebra over modules (for more details on this monad see [13]).

Here we refer to additive categories, with a monoidal coalgebra modality, as "additive linear categories": their biproduct completion is then an "additive monoidal storage category" [4]. This latter was the setting being considered by Fiore in [7]. An additive monoidal storage category (which has biproducts) is always an additive linear category (which may not have biproducts) and both always have a canonical monoidal bialgebra modality. Here, to facilitate the proofs, it is convenient to work with a further intermediate notion we called an "additive bialgebra modality": this is a bialgebra modality which has an additional coherence requirement between the additive and bialgebra structure. It is with respect to this structure that we prove that deriving transformations and coderelictions are equivalent.

Coderelictions always give deriving transformations, and it was known [2] that a deriving transformation, for a bialgebra modality, is equivalent to a codereliction if and only if the deriving transformation satisfies the ∇-rule. However, this latter rule was thought to be a completely independent requirement. The key new observation is that, for an additive bialgebra modality with a deriving transformation the ∇-rule is, in fact, implied. More specifically, while a deriving transformation is assumed to satisfy five rules – **[d.1]**–**[d.5]** below – which include the linear rule and the Leibnitz rule, over an additive bialgebra modality, we prove that, for a deriving transformation which satisfies just the linear rule, the ∇-rule is equivalent to the Leibniz rule.

When an additive symmetric monoidal category has a monoidal coalgebra modality it is straightforward to show that the modality is an additive bialgebra modality. Thus, for additive linear categories deriving transformations and coderelictions are also equivalent.

Clearly additive bialgebra modalities and monoidal coalgebra modalities are closely related. In particular, additive bialgebra modalities can always be extended to the biproduct completion (see Appendix C) and, furthermore, this biproduct completion has Seely isomorphisms [11]. Thus, additive bialgebra modalities correspond to monoidal storage categories [4] (also called new Seely categories [1, 10]) having the Seely isomorphisms. However, it is well-known (as the name suggests) that monoidal storage categories have a monoidal coalgebra modality.

Thus, additive bialgebra modalities are, in fact, monoidal coalgebra modalities.

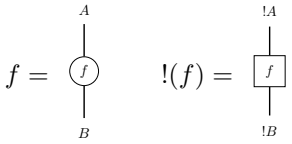
To complete the main story of the paper, we observe that in an additive linear category coderelictions and deriving transformations always satisfy the "strength laws": these were the subject of an addendum added to [7]. We provide a proof in the setting of additive linear categories. Putting all this together one has that – contrary to the suggestion in [7] – deriving transformations and creation operators are, in additive linear categories, completely equivalent.

2 Conventions and the Graphical Calculus

We shall use diagrammatic order for composition: explicitly, this means that the composite map fg is the map which first does f then g. Furthermore, to simplify working in symmetric monoidal categories, we will allow ourselves to work in strict symmetric monoidal categories and so will generally suppress the associator and unitor isomorphisms. For a symmetric monoidal category we will use $\otimes$ for the tensor product, K for the unit, and $\sigma : A \otimes B \to B \otimes A$ for the symmetry isomorphism.

We shall make extensive use of the graphical calculus [8] for symmetric monoidal categories as this makes proofs easier to follow. We refer the reader to [12] for an introduction to the graphical calculus in monoidal categories and its variations. Note, however, that our diagrams are to be read down the page – from top to bottom – and we shall often omit labelling wires with objects.

We will be working with coalgebra modalities: these are based on a comonad $(!, \delta, \varepsilon)$ where $!$ is the functor, δ is the comultiplicaton and ε is the counit. As in [2], we will use functor boxes when dealing with string diagrams involving the functor $!$: a mere map $f : A \to B$ will be encased in a circle while $!(f) : !A \to !B$ will be encased in a box:



3 Differential Categories

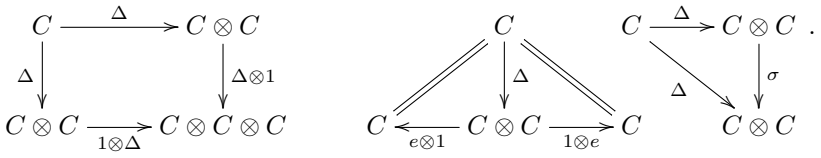
Tensor differential categories are structures over additive symmetric monoidal categories with a coalgebra modality. We begin by recalling the components of this structure starting with the notion of an additive category. Here we mean "additive" in the sense of being commutative monoid enriched: we do not assume negatives nor do we assume biproducts (this differs from the usage in [9] for example). This allows many important examples such as the category of sets and relation or the category of modules for a commutative rig[1].

▶ **Definition 1.** An **additive category** is a commutative monoid enriched category, that is, a category in which each hom-set is a commutative monoid with an addition operation $+$ and a zero 0, and such that composition preserves the additive structure, that is: $k(f+g)h = kfh+kgh$, $0f = 0$ and $f0 = 0$. An **additive symmetric monoidal category** is a symmetric monoidal category which is also an additive category in which the tensor product is compatible

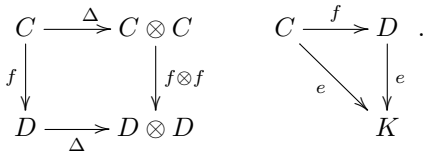
[1] Rigs are also known as a semirings: they are ri*n*gs without *n*egatives.

with the additive structure in the sense that $k \otimes (f+g) \otimes h = k \otimes f \otimes h + k \otimes g \otimes h$, $0 \otimes h = 0$, and $h \otimes 0 = 0$.

In a symmetric monoidal category, a cocommutative comonoid is a triple (C, Δ, e) consisting of an object C, a map $\Delta : C \to C \otimes C$ called the comultiplication and a map $e : C \to K$ called the counit such that the following diagrams commute:



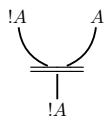
A morphism of comonoids $f : (C, \Delta, e) \to (D, \Delta', e')$ is a map $f : C \to D$ which preserves the comultiplication and counit, that is, the following diagrams commute:



▶ **Definition 2.** A **coalgebra modality** [2] on a symmetric monoidal category is a quintuple $(!, \delta, \varepsilon, \Delta, e)$ consisting of a comonad $(!, \delta, \varepsilon)$, a natural transformation Δ with components $\Delta_A : !A \to !A \otimes !A$, and a natural transformation e with components $e_A : !A \to K$ such that for each object A, $(!A, \Delta_A, e_A)$ is a cocommutative comonoid and δ preserves the comultiplication, that is, $\delta\Delta = \Delta(\delta \otimes \delta)$.

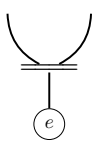
One can prove that δ also preserves the counit, $\delta e = e$, and so δ is actually a comonoid homomorphism. Furthermore, requiring that Δ and e be natural transformations is equivalent to asking that for each map $f : A \to B$, $!(f) : !A \to !B$ is a comonoid morphism. When combined with the additive structure, this ensures that $!A$ is a coalgebra in the classical algebraic sense. Note that, for now, we do not assume that the functor $!$ of a coalgebra modality is a monoidal functor (this will come later). The coKleisli maps for the comonad are important: these maps are of the form $f : !A \to B$, which amongst these are the **linear maps** $\varepsilon g : !A \to B$ where $g : A \to B$.

▶ **Definition 3.** A **differential category** is an additive symmetric monoidal category with a coalgebra modality which comes equipped with a **deriving transformation** [2], that is, a natural transformation d with components $\mathsf{d}_A : !A \otimes A \to !A$, which is represented in the graphical calculus as:

$\mathsf{d} :=$ 

such that d satisfies the following equations:

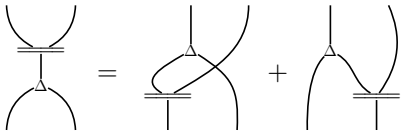
[d.1] Constant Rule: $\mathsf{d}e = 0$



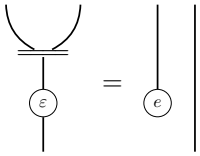
$= 0$

[d.2] Leibniz Rule:

$$\mathsf{d}\Delta = (\Delta \otimes 1)(1 \otimes \sigma)(\mathsf{d} \otimes 1) + (\Delta \otimes 1)(1 \otimes \mathsf{d})$$

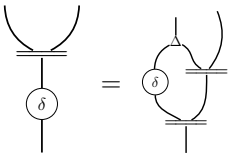


[d.3] Linear Rule: $\mathsf{d}\varepsilon = e \otimes 1$

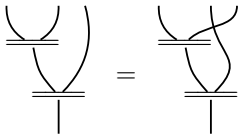


[d.4] Chain Rule:

$$\mathsf{d}\delta = (\Delta \otimes 1)(\delta \otimes 1 \otimes 1)(1 \otimes \mathsf{d})\mathsf{d}$$



[d.5] Interchange Rule: $(\mathsf{d} \otimes 1)\mathsf{d} = (1 \otimes \sigma)(\mathsf{d} \otimes 1)\mathsf{d}$



The first axiom **[d.1]** states that the derivative of a constant map is zero. The second axiom **[d.2]** is the Leibniz rule or the product rule for differentiation. The third axiom **[d.3]** says that the derivative of a linear map is constant. The fourth axiom **[d.4]** is the chain rule. The last axiom **[d.5]** is the interchange law, which naively states that differentiating with respect to x then y is the same as differentiation with respect to y then x. It should be noted that **[d.5]** was not originally a requirement in [2] but was later added to the definition to ensure that the coKleisli category of a differential category was a Cartesian differential category [3]. It should also be noted that by the naturality of e and d, the constant rule **[d.1]** is in fact derivable:

▶ **Lemma 4.** *For a coalgebra modality on additive symmetric monoidal category, any natural transformation* $\mathsf{d}_A : !A \otimes A \to !A$ *satisfies the constant rule* ***[d.1]****.*

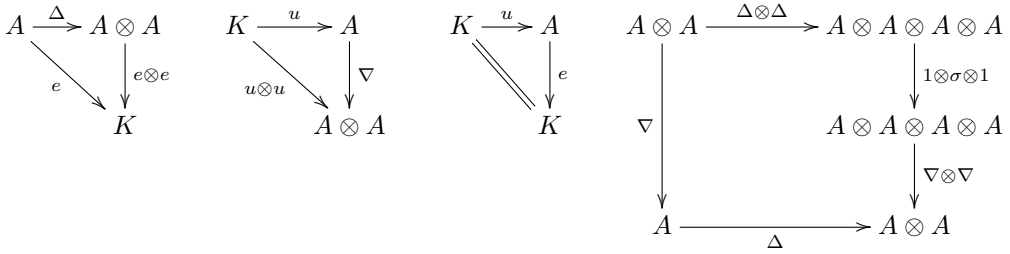
Proof. By naturality and the additive structure: $\mathsf{d}e = \mathsf{d}!(0)e = (!(0) \otimes 0)\mathsf{d}e = 0\mathsf{d}e = 0$. ◀

Therefore a deriving transformation is simply a natural transformation which satisfies the Leibniz rule **[d.2]**, the linear rule **[d.3]**, the chain rule **[d.4]** and the interchange rule **[d.5]**.

4 Bialgebra Modalities, coderelictions and the ∇-Rule

In [2], Blute, Cockett and Seely observed that if the coalgebra modality came equipped with a bialgebra structure and a codereliction then one could obtain a deriving transformation. We recall these ideas, starting with the notion of a bialgebra modality. In a symmetric monoidal

category, a (commutative) bialgebra is a quintuple $(A, \nabla, u, \Delta, e)$ such that (A, ∇, u) is a commutative monoid, (A, Δ, e) is a cocommutative comonoid, and the following diagrams commute:



▶ **Definition 5.** A **bialgebra modality** [2] on an additive symmetric monoidal category is a coalgebra modality $(!, \delta, \varepsilon, \Delta, e)$ equipped with a natural transformation $\nabla : !A \otimes !A \to !A$, and a natural transformation $u : K \to !A$ such that $(!A, \nabla, u, \Delta, e)$ is a bialgebra for each object A, and ε is compatible with ∇ in the following sense: $\nabla\varepsilon = \varepsilon \otimes e + e \otimes \varepsilon$.

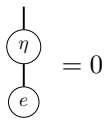
By the naturality of ∇, Δ, u and e we note that for every map f, $!(f)$ is both a monoid and comonoid morphism. Also, in the original definition of a bialgebra modality in [2], it was also required that ε be compatible with u, however this compatibility is provable by the naturality of u and ε. The proof is similar to that of Lemma 4.

▶ **Lemma 6.** *For a bialgebra modality* $u\varepsilon = 0$.

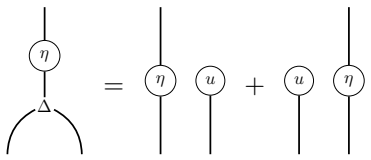
The key ingredient required to obtain a differential for a bialgebra modality is:

▶ **Definition 7.** A **codereliction** [2] for a bialgebra modality on an additive symmetric monoidal category is a natural transformation $\eta_A : A \to !A$, such that the following equalities hold:

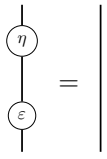
[dC.1] Constant Rule: $\eta e = 0$



[dC.2] Product Rule: $\eta\Delta = \eta \otimes u + u \otimes \eta$

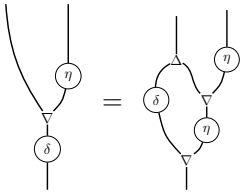


[dC.3] Linear Rule: $\eta\varepsilon = 1$



[dC.4] Chain Rule:

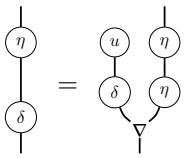
$$(1 \otimes \eta)\nabla\delta = (\Delta \otimes \eta)(1 \otimes \nabla)(\delta \otimes \eta)\nabla$$



As for the constant rule for the deriving transformation, the constant rule **[dC.1]** for a codereliction can be derived (the proof is similar to that of Lemma 4):

▶ **Lemma 8.** *For a bialgebra modality, any natural transformation $\eta_A : A \to !A$ satisfies the constant rule **[dC.1]**.*

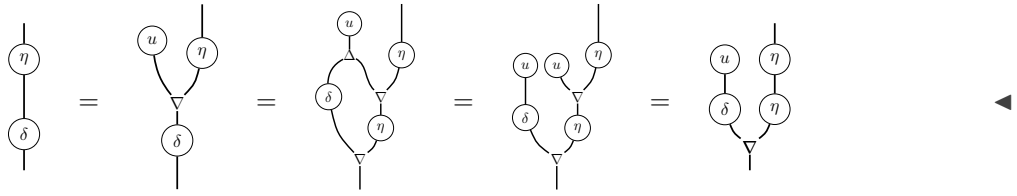
Thus, a codereliction is simply a natural transformation which satisfies the product rule **[dC.2]**, the linear rule **[dC.3]** and the chain rule **[dC.4]**. In [7], Fiore uses an alternative axiom for the chain rule **[dC.4]** which is stated as follows:
[dC.4'] Alternative Chain Rule: $\nabla\delta = (u \otimes \eta)(\delta \otimes \eta)\nabla$



Although it was not noted in [7], in that setting both **[dC.4]** and **[dC.4′]** are equivalent. In the setting of a mere bialgebra modality, it is clear that **[dC.4]** implies **[dC.4′]**: the reverse implication, however, does not appear to hold. Thus, at this stage we prove the implication in one direction:

▶ **Lemma 9.** *Any natural transformation which satisfies the chain rule **[dC.4]** also satisfies the alternative chain rule **[dC.4′]**.*

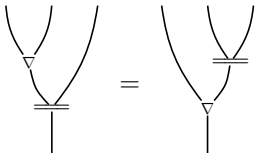
Proof. The bialgebra structure gives the following chain of equalities:



◀

Let us now consider the relation between deriving transformations and bialgebra modalities. This is captured by the ∇-rule [2]:

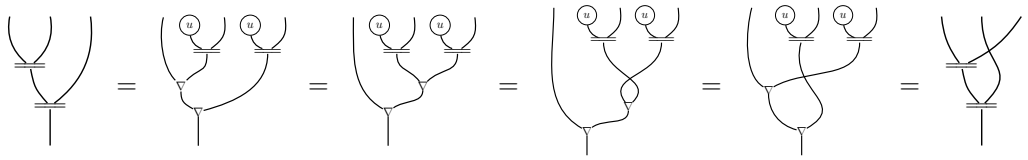
▶ **Definition 10.** For a bialgebra modality, a natural transformation $\mathsf{d}_A : !A \otimes A \to !A$ is said to satisfy the **∇-rule** if:
[d.∇] ∇-Rule: $(\nabla \otimes 1)\mathsf{d} = (1 \otimes \mathsf{d})\nabla$



Notice this implies that $\mathsf{d} = (1 \otimes u \otimes 1)(\nabla \otimes 1)\mathsf{d} = (1 \otimes u \otimes 1)(1 \otimes \mathsf{d})\nabla$. Thus, an immediate consequence of satisfying the ∇-rule is interchange rule:

▶ **Lemma 11.** *For a bialgebra modality, a natural transformation* d *which satisfies the* ∇*-rule, [**d.**∇], also satisfies the interchange rule, [**d.5**].*

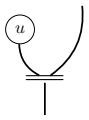
Proof. By [**d.**∇] and the commutative monoid structure, we have the following equality:



◀

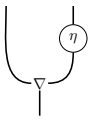
Therefore, for a bialgebra modality, a natural transformation which satisfies the Leibniz rule [**d.2**], the linear rule [**d.3**], the chain rule [**d.4**], and [**d.**∇] is a deriving transformation. Furthermore, all deriving transformations which satisfy the ∇-rule [**d.**∇] induce a codereliction defined as:

$$eta := \; A \xrightarrow{u \otimes 1} !A \otimes A \xrightarrow{\mathsf{d}} !A \qquad \eta := \;$$



Conversely, every codereliction induces a deriving transformation which satisfies the ∇-rule:

$$\mathsf{d} := \; !A \otimes A \xrightarrow{1 \otimes \eta} !A \otimes !A \xrightarrow{\nabla} !A \qquad \mathsf{d} := \;$$



Using the same proof in [2] – which was for monoidal storage categories – it is easily seen that:

▶ **Theorem 12.** *For an additive symmetric monoidal category with a bialgebra modality, deriving transformations, which satisfy the* ∇*-rule [**d.**∇], are in bijective correspondence to coderelictions by:*

$$\mathsf{d} \longmapsto \eta := (u \otimes 1)\mathsf{d}\,, \qquad \eta \longmapsto \mathsf{d} := (1 \otimes \eta)\nabla\,.$$

5 Additive Bialgebra Modalities

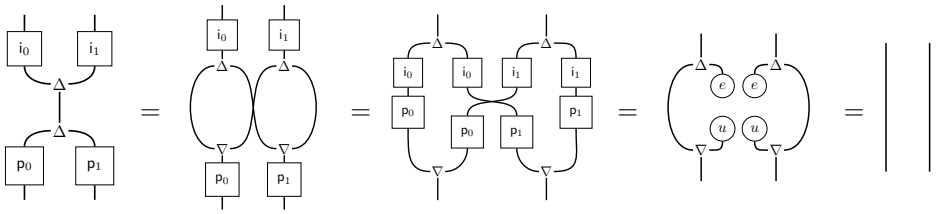
In this section we introduce the concept of an *additive* bialgebra modality. Our goal will be to show that a deriving transformation for an additive bialgebra modality induces a codereliction. Since a codereliction always implies a deriving transformation, this shows that, for an additive bialgebra modality, deriving transformations and codericton maps are equivalent. A source of examples of additive bialgebra modalities will be derived in the next section. Additive bialgebra modalities require additional coherence with the additive structure:

▶ **Definition 13.** An **additive bialgebra modality** on an additive symmetric monoidal category is a bialgebra modality which is compatible with the additive structure in the sense that $!(0) = eu$ and $!(f + g) = \Delta(!(f) \otimes !(g))\nabla$.

First we examine coderelictions for additive bialgebra modalities. When a natural transformations η is a section of ε, that is, η satisfies **[dC.3]**, we can define four natural transformations: $\mathsf{p}_0 = \varepsilon \otimes e$, $\mathsf{p}_1 = e \otimes \varepsilon$, $\mathsf{i}_0 = \eta \otimes u$, $\mathsf{i}_1 = u \otimes \eta$. Notice that as η satisfies the constant rule **[dC.1]** and the linear rule **[dC.3]** and from the properties of a bialgebra modality that we have: $\mathsf{i}_j\mathsf{p}_k = 0$ when $j \neq k$ and $\mathsf{i}_j\mathsf{p}_j = 1$, which is reminiscent of the identities satisfied by the projection and injection maps of a biproduct. These maps will be key to the proof of Lemma 15 below. For an additive bialgebra modality this means that we have: $!(\mathsf{i}_j)!(\mathsf{p}_k) = eu$ when $j \neq k$ and $!(\mathsf{i}_j)!(\mathsf{p}_j) = 1$. This allows the derivation of the following useful identity:

▶ **Lemma 14.** *For an additive bialgebra modality and a natural transformations η which satisfies the linear rule **[dC.3]**:* $(!(\mathsf{i}_0) \otimes !(\mathsf{i}_1))\nabla\Delta(!(\mathsf{p}_0) \otimes !(\mathsf{p}_1)) = 1$.

Proof. By the bialgebra modality, we have the following equality:



◀

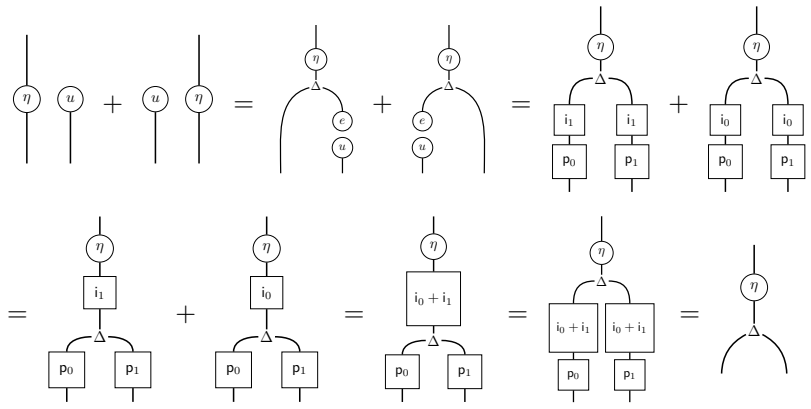
Furthermore, for an additive bialgebra modality, the linear rule **[dC.3]** implies the product rule **[dC.2]**:

▶ **Lemma 15.** *For an additive bialgebra modality, any natural transformation, η, which satisfies the linear rule, **[dC.3]**, also satisfies the product rule, **[dC.2]**.*

Proof. Notice that by naturality of η, we have that:

$$\eta!(f+g) = (f+g)\eta = f\eta + g\eta = \eta!(f) + \eta!(g)\,.$$

Then, using i_j and p_k, we have the following:



◀

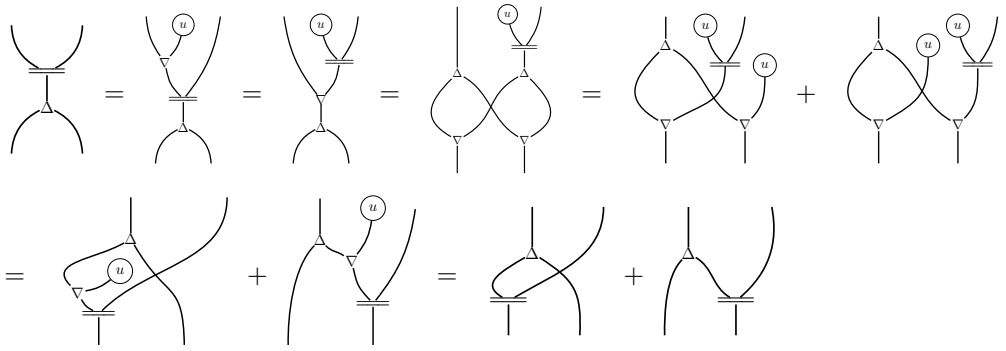
Therefore, for an additive bialgebra modality, a natural transformation which simply satisfies the linear rule **[dC.3]** and the chain rule **[dC.4]** is a codereliction.

Turning our attention to deriving transformations for additive bialgebra modalities, we begin by noticing that satisfying the Leibniz rule is equivalent to satisfying the ∇-rule:

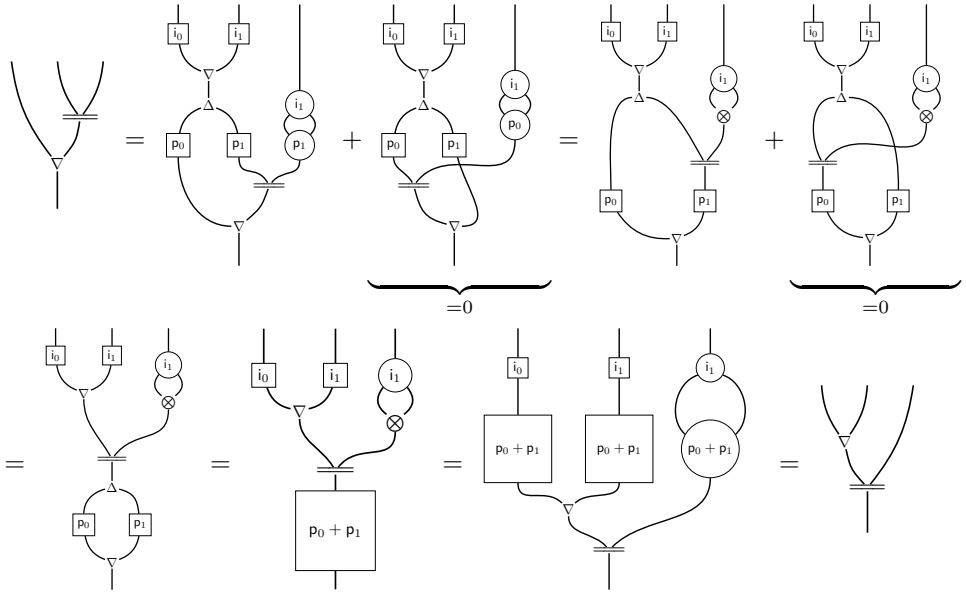
▶ **Theorem 16.** *For an additive bialgebra modality and a natural transformation* d *satisfying the linear rule* ***[d.3]****, the Leibniz rule* ***[d.2]*** *is satisfied if and only if the* ∇*-rule* ***[d.*∇*]*** *is satisfied.*

Proof.

[d.∇] ⇒ [d.2]: It is easy to see that since d satisfies [**d.3**] that $(u \otimes 1)\mathsf{d}$ satisfies [**dC.3**], the linear rule for coderelictions. However, by Lemma 15, this implies that $(u \otimes 1)\mathsf{d}$ satisfies [**dC.2**], the product rule for coderelictions. Therefore, we have:



[d.2] ⇒ [d.∇]: By the properties of i_j and p_k, and the identity of Lemma 14, we have:



◀

Therefore, we obtain the following theorem:

▶ **Theorem 17.** *For an additive bialgebra modality, every deriving transformation satisfies the* ∇*-rule* ***[d.*∇*]****, thus is induced equivalently by a coderelictions.*

6 Additive Linear Categories

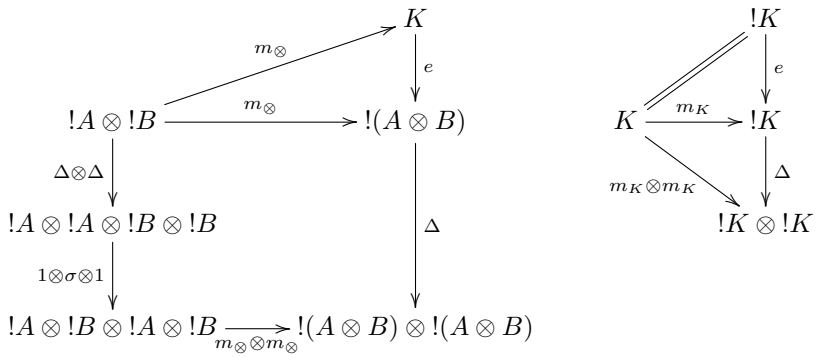
We now turn our attention to the case when the coalgebra modality is monoidal. Recall that a symmetric monoidal functor [9] is a functor ! equipped with a natural transformation

$m_\otimes : !A \otimes !B \to !(A \otimes B)$ and a map $m_K : K \to !K$ satisfying certain coherences. This can be extended to defining a symmetric monoidal comonad by asking that δ and ε be monoidal natural transformations. For a full detailed list of the coherences for symmetric monoidal comonads, see [1]. The string diagrams representations of $m_\otimes$ and m_K are as follows:

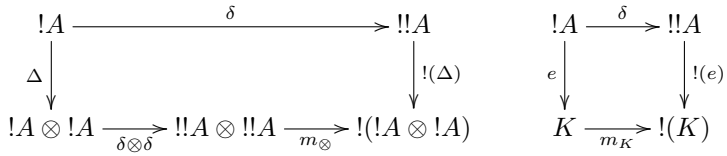
$m_\otimes =$  $m_K =$ 

▶ **Definition 18.** A **monoidal coalgebra modality** on a symmetric monoidal category is a symmetric monoidal comonad, $(!, \delta, \varepsilon, m_\otimes, m_K)$, and a coalgebra modality, $(!, \delta, \varepsilon, \Delta, e)$, satisfying

(i) Δ and e are monoidal transformations, that is, the following diagrams commute:



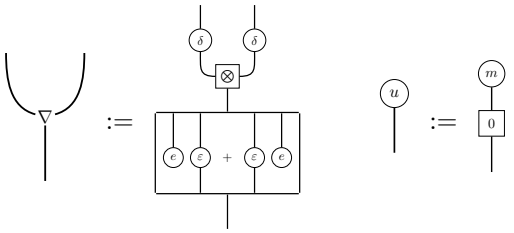
(ii) Δ and e are !-coalgebra morphisms, that is, the following diagrams commute:



A **linear category** [1, 4] is a symmetric monoidal category with a monoidal coalgebra modality; an **additive linear category** is a linear category which is also an additive symmetric monoidal category.

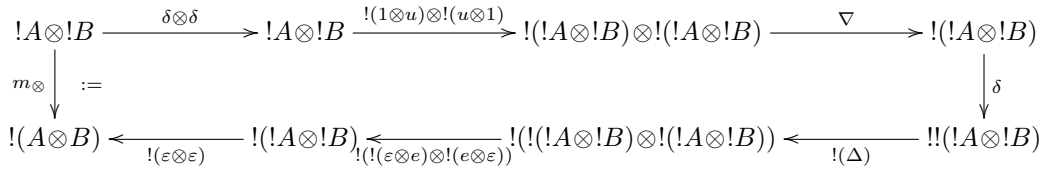
A monoidal coalgebra modality on an additive linear category induces an additive bialgebra modality. The multiplication and unit are respectively:

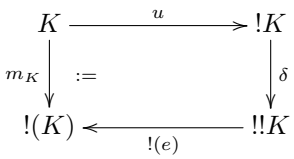
$$!A \otimes !A \xrightarrow{\delta\otimes\delta} !!A \otimes !!A \xrightarrow{m_\otimes} !(!A \otimes !A) \xrightarrow{!(\varepsilon\otimes e + e\otimes\varepsilon)} !A \qquad K \xrightarrow{m_K} !K \xrightarrow{!(0)} !A$$



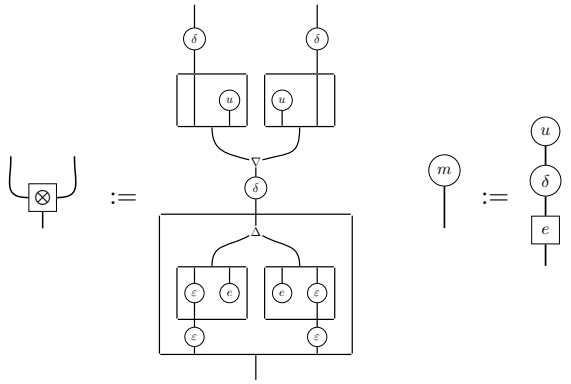
The converse statement is also, in fact, true: an additive bialgebra modality induces a monoidal coalgebra modality. The monoidal structure $m_\otimes$ and m_K are defined respectively

as:





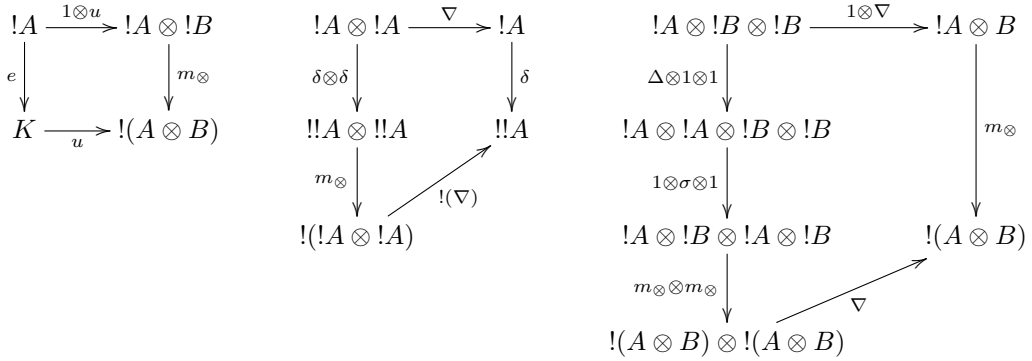
Represented in string diagrams as:



▶ **Theorem 19.** *For an additive symmetric monoidal category, monoidal coalgebra modalities correspond bijectively to additive bialgebra modalities.*

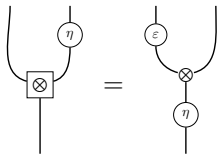
An indirect proof of this is discussed at the end of the introduction: a direct proof is long and quite technical. The identities expressing compatibility between the bialgebra modality and the monoidal coalgebra which were observed in [7] (Theorem 3.1) to hold in an additive monoidal storage category also hold in an additive linear category (see Appendix A for a proof):

▶ **Lemma 20.** *In an additive linear category, the following diagrams commute:*



We now turn our attention to the relation between the monoidal structure and the differential structure. In [7] Fiore introduced another axiom relating η to the monoidal structure:

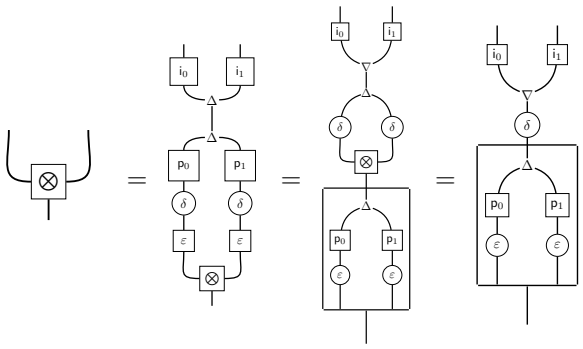
[dC.m] Monoidal Rule: $(1 \otimes \eta)m_{\otimes} = (\varepsilon \otimes 1)\eta$



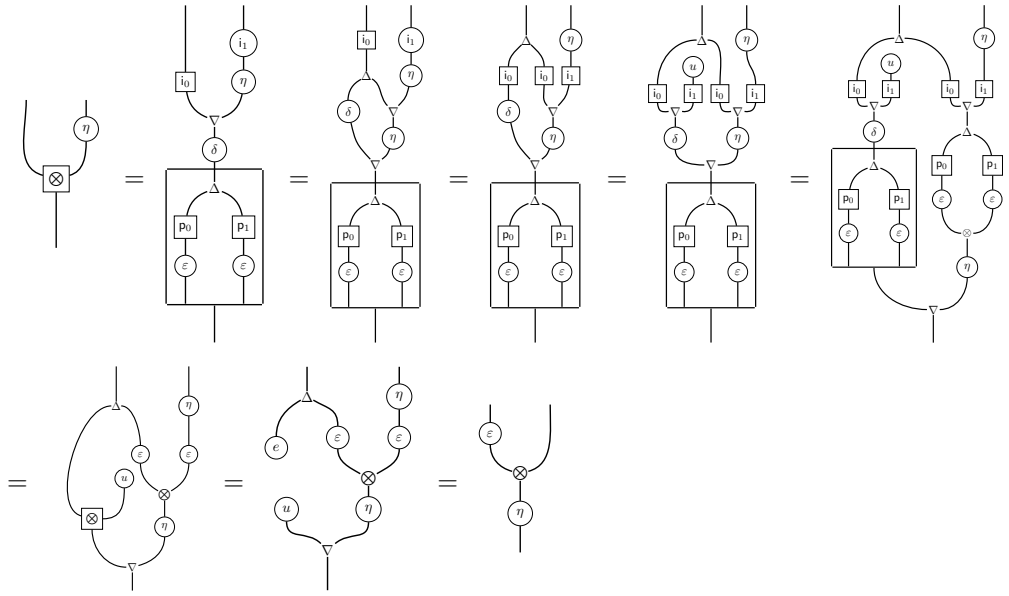
It turns out that coderelictions for the induced additive bialgebra modality always satisfy the monoidal rule **[dC.m]**:

▶ **Lemma 21.** *In an additive linear category, coderelictions on the induced additive bialgabra modality satisfy the monoidal rule **[dC.m]**.*

Proof. By Lemma 14 and the fact that Δ is a !-coalgebra morphism, we first have that:



where i_j and p_k are defined as in the previous section. Expressing $m_{\otimes}$ as above, then by the linear rule **[dC.3]**, chain rule **[dC.4]**, the naturality of u and ∇, and the first diagram of Lemma 20 we have the following equality :

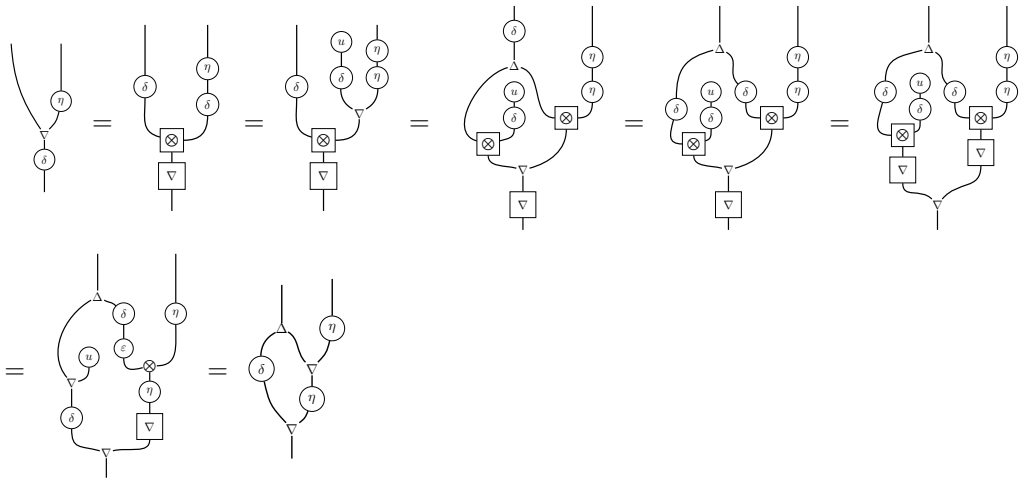


◀

Conversely, the alternative chain rule **[dC.4′]** and the monoidal rule **[dC.m]** imply the chain rule **[dC.4]**.

▶ **Lemma 22.** *In an additive linear category, a natural transformation which satisfies the alternative chain rule **[dC.4′]** and the monoidal rule **[dC.m]**, on the the induced additive bialgebra modality, also satisfies the chain rule **[dC.4]**.*

Proof. By the compatiblity relations of Lemma 20 and the coalgebra modality requirements we have:



◀

▶ **Corollary 23.** *For an additive linear category the following are equivalent for a natural transformation η and the induced additive bialgebra modality:*

(i) *η is a codereliction;*

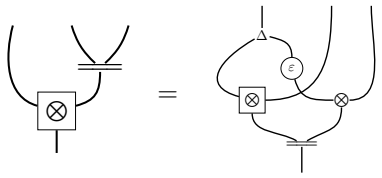
(ii) *η satisfies the linear rule **[dC.3]** and the chain rule **[dC.4]**;*

(iii) *η satisfies the linear rule **[dC.3]**, the alternative chain rule **[dC.4′]** and the monoidal rule **[dC.m]**.*

Part (iii) is the definition of Fiore's creation map: this shows that the original definition of a codereliction is equivalent to Fiore's creation map.

Finally we explore deriving transformations of additive linear categories. The compatibility between a deriving transformation and the monoidal structure is described by the monoidal rule – this is the strength rule which was the subject of Fiore's addendum:

[d.m] Monoidal Rule: $(1 \otimes \mathsf{d})m_{\otimes} = (\Delta \otimes 1 \otimes 1)(1 \otimes \varepsilon \otimes 1 \otimes 1)(1 \otimes \sigma \otimes 1)(m_{\otimes} \otimes 1 \otimes 1)\mathsf{d}$



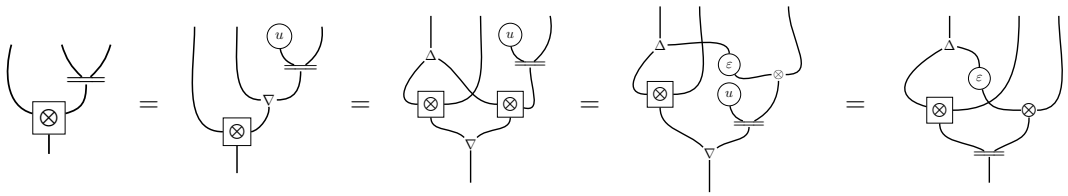
Fiore's creation operator [7] was defined to satisfy the linear rule **[d.3]**, the chain rule **[d.4]**, the ∇-rule **[d.∇]**, and the monoidal rule **[d.m]** – in his addendum, he pointed out the latter was redundant. It turns out that when a natural transformation satisfies both the linear rule **[d.3]** and the chain rule **[d.4]**, then the monoidal rule is equivalent to both the ∇-rule and the Leibniz rule:

▶ **Theorem 24.** *For the induced additive bialgebra modality of an additive linear category and a natural transformation satisfying the linear rule **[d.3]** and the chain rule **[d.4]**, the following are equivalent:*

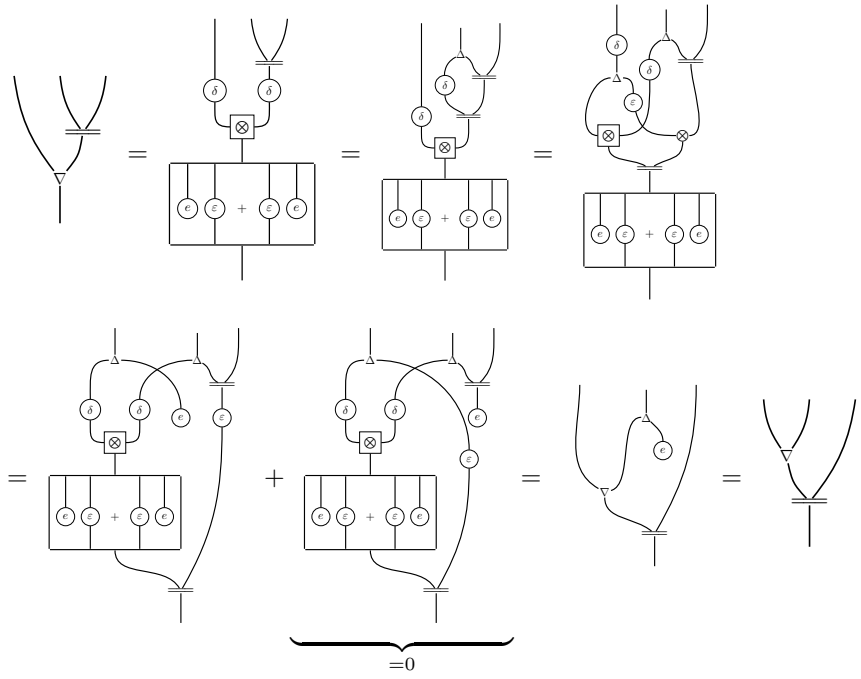
(i) *the Leibniz rule* ***[d.2]****;*
(ii) *the* ∇*-rule* ***[d.*∇*]****;*
(iii) *the monoidal rule* ***[d.m]***

Proof. Since this is an extension of Lemma 16, it suffices to show that the ∇-rule [**d.**∇] and the monoidal rule [**d.m**] are equivalent.

[d.∇**] ⇒ [d.m]:** It is easy to see that since d satisfies the linear rule [**d.3**] and the chain rule [**d.4**], $(u \otimes 1)\mathsf{d}$ satisfies the codereliction linear rule [**dC.3**] and chain rule [**dC.4**], and therefore by Corollary 23 is a codereliction and which by Lemma 21 satisfies the codereliction monoidal rule [**dC.m**]. Therefore, by the compatibility relations of Lemma 20, we have:



[d.m] ⇒ [d.∇**]:** By expanding ∇ using $m_{\otimes}$, we obtain the following equality:



◀

Finally, this gives the following theorem:

▶ **Theorem 25.** *For the induced additive bialgebra modality of an additive linear category, all deriving transformations satisfy the monoidal rule* ***[d.m]*** *and are induced by a codereliction.*

7 Conclusion

Of course it would have been very much simpler if all these observations had been made in [2], the original paper. Unfortunately, they were not. The suggestion in Marcelo Fiore's

paper [7] that something new and different had been found, made us realize that the relation between deriving transformations and coderelictions had never been fully explored. Here we have revisited this relation. Significantly, it is not that the notion of differentiation varies but rather, as the setting varies, significantly different presentations of the differential become possible. Philosophically this is certainly how thing should be ... and apparently it is how they are!

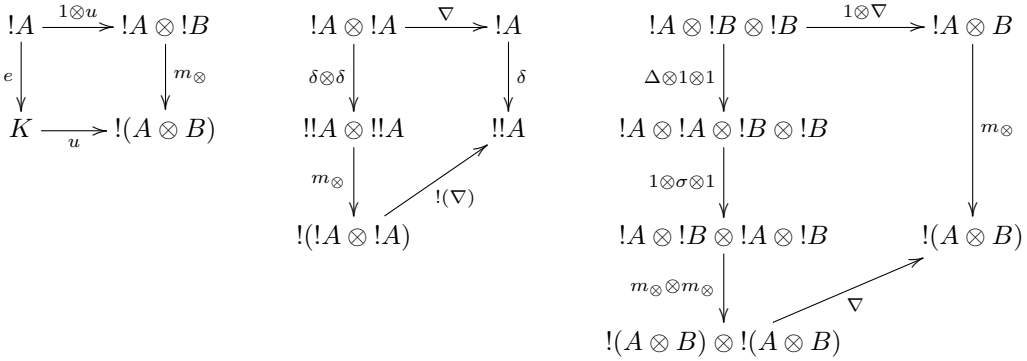
References

1. Gavin Bierman. What is a categorical model of intuitionistic linear logic? *Typed Lambda Calculi and Applications*, pages 78–93, 1995.
2. Richard F. Blute, J. Robin B. Cockett, and Robert A. G. Seely. Differential categories. *Mathematical structures in computer science*, 16(06):1049–1083, 2006.
3. Richard F. Blute, J. Robin B. Cockett, and Robert A. G. Seely. Cartesian differential categories. *Theory and Applications of Categories*, 22(23):622–672, 2009.
4. Richard F. Blute, J. Robin B. Cockett, and Robert A. G. Seely. Cartesian differential storage categories. *Theory and Applications of Categories*, 30(18):620–686, 2015.
5. Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1):1–41, 2003.
6. Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Theoretical Computer Science*, 364(2):166–195, 2006.
7. Marcelo P. Fiore. Differential structure in models of multiplicative biadditive intuitionistic linear logic. In *International Conference on Typed Lambda Calculi and Applications*, pages 163–177. Springer, 2007.
8. André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, 1991.
9. Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 2013.
10. Paul-André Mellies. Categorical models of linear logic revisited, 2003.
11. Robert A. G. Seely. *Linear logic,*-autonomous categories and cofree coalgebras*. Ste. Anne de Bellevue, Quebec: CEGEP John Abbott College, 1987.
12. Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010.
13. Shilong Zhang, Li Guo, and William Keigher. Monads and distributive laws for rota–baxter and differential algebras. *Advances in Applied Mathematics*, 72:139–165, 2016.

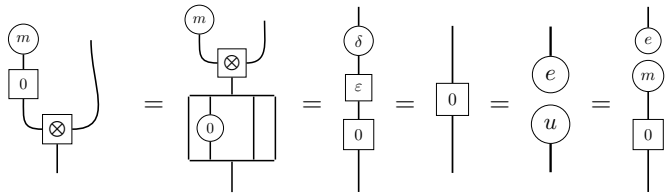
A Compatibilities of the Bialgebra Modality

We provide the proof of some compatibility relations between the bialgebra modality and the monoidal structure:

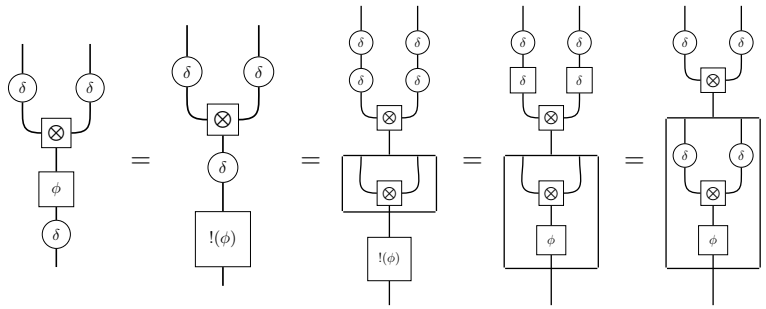
▶ **Lemma 26.** *In an additive linear category, the following diagrams commute:*



Proof. For the first square on the left we use the naturality of $m_\otimes$, the unit laws of the monoidal functor and that $!(0)$ splits:

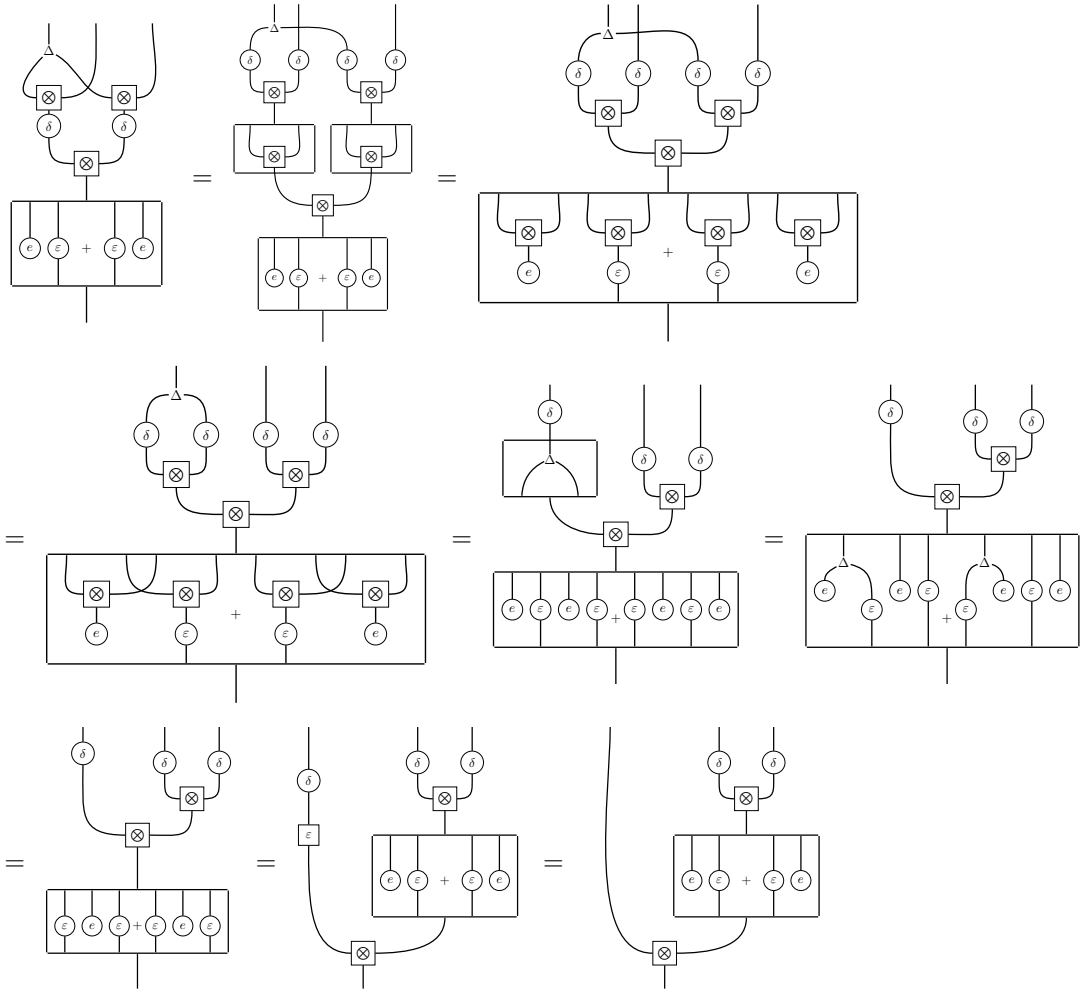


Now, for space and simplification, define $\phi = e \otimes \varepsilon + \varepsilon \otimes e$. Then for the middle square, we use the naturality of $m_\otimes$, the comonad δ square and that δ is a monoidal transformation:



For the square on the right we use that δ is a monoidal transformation, naturality of $m_\otimes$, associativity and symmetry of $m_\otimes$, that e and ε are monoidal transformations and that Δ is

a !-coalgebra morphism:



◀

B Seely Isomorphisms

In a symmetric monoidal category with finite products $\times$ and terminal object T, a coalgebra modality has **Seely isomorphisms** [1, 4] if the natural transformations χ and χ_{T} defined respectively as:

$$!(A \times B) \xrightarrow{\Delta} !(A \times B) \otimes !(A \times B) \xrightarrow{!(\pi_0)\otimes !(\pi_1)} !A \otimes !B \qquad !(\mathsf{T}) \xrightarrow{e} K$$

are isomorphisms, so $!(A \times B) \cong !A \otimes !B$ and $!(\mathsf{T}) \cong K$.

▶ **Definition 27.** A **monoidal storage category** [4] is a symmetric monoidal category with finite products and a coalgebra modality which has Seely isomorphisms.

Monoidal storage categories were called **new Seely categories** in [1, 10]. As explained in [4], every coalgebra modality which satisfies the Seely isomorphisms is a monoidal coalgebra modality, where $m_{\otimes}$ is

$$!A \otimes !B \xrightarrow{\chi^{-1}} !(A \times B) \xrightarrow{\delta} !!(A \times B) \xrightarrow{!(\chi)} !(!A \otimes !B) \xrightarrow{!(\varepsilon\otimes\varepsilon)} !(A \otimes B)$$

and m_K is defined as

$$K \xrightarrow{\chi_{\mathsf{T}}^{-1}} !(\mathsf{T}) \xrightarrow{\delta} !!(\mathsf{T}) \xrightarrow{!(\chi_{\mathsf{T}})} !(K)$$

Conversely, in the presence of finite products, every monoidal coalgebra modality satisfies the Seely isomorphisms [1] where the inverse of χ is

$$!A \otimes !B \xrightarrow{\delta\otimes\delta} !!A \otimes !!B \xrightarrow{m_\otimes} !(!A \otimes !B) \xrightarrow{!(\langle \varepsilon\otimes e, e\otimes\varepsilon\rangle)} !(A \times B)$$

while the inverse of χ_{T} is

$$K \xrightarrow{m_K} !(K) \xrightarrow{!(\mathsf{t})} !(\mathsf{T})$$

where $\mathsf{t} : K \to \mathsf{T}$ is the unique map to the terminal object. Therefore we obtain the following theorem (Theorem 3.1.6 [4]):

▶ **Theorem 28.** *Every monoidal storage category is a linear category and conversely, every linear category with finite products is a monoidal storage category.*

This together with Appendix C provides a rather indirect verification of Theorem 19.

We now turn our attention to monoidal storage categories with an additive structure:

▶ **Definition 29.** An **additive monoidal storage category** is a monoidal storage category which is also an additive symmetric monoidal category.

Notice, this implies that additive monoidal storage categories have finite biproducts $\times$ and a zero object 0. As noted in [2], the coalgebra modality of an additive monoidal storage category is an additive bialgebra modality where the multiplication and unit are defined respectively as:

$$!A \otimes !A \xrightarrow{\chi^{-1}} !(A \times A) \xrightarrow{!(\nabla_\times)} !(A) \qquad K \xrightarrow{\chi_0^{-1}} !0 \xrightarrow{!(0)} !A$$

Conversely, every additive bialgebra modality satisfies the Seely isomorphisms where χ^{-1} and χ_0^{-1} are defined respectively as:

$$!A \otimes !B \xrightarrow{!(\iota_0)\otimes!(\iota_1)} !(A \times B) \otimes !(A \times B) \xrightarrow{\nabla} !(A \times B) \qquad K \xrightarrow{u} !0$$

It is easy to check that this indeed gives the Seely isomorphisms, therefore we have:

▶ **Theorem 30.** *The following are equivalent:*

(i) *An additive monoidal storage category;*
(ii) *An additive linear category with finite biproducts;*
(iii) *An additive symmetric monoidal category with finite biproducts and an additive bialgebra modality.*

C Biproduct Completion

Every additive bialgebra modality induces an additive monoidal storage category the biproduct completion. We first recall the biproduct completion for an additive category [9].

Let $\mathbb{X}$ be an additive category. Define the biproduct completion of $\mathbb{X}$, $\mathsf{B}[\mathbb{X}]$, as the category whose objects are list of objects of $\mathbb{X}$: $(A_1, \ldots, A_n)$, including the empty list $()$, and whose maps are matrices of maps of $\mathbb{X}$, including the empty matrix:

$$(A_1, \ldots, A_n) \xrightarrow{[f_{i,j}]} (B_1, \ldots, B_m)$$

where $f_{i,j} : A_i \to B_j$. The composition in $\mathsf{B}[\mathbb{X}]$ is the standard matrix multiplication:

$$[f_{i,j}][g_{l,k}] = [\sum f_{i,k} g_{k,j}]$$

while the identity is the standard identity matrix:

$$(A_1, \ldots, A_n) \xrightarrow{[\delta_{i,j}]} (A_1, \ldots, A_n)$$

where $\delta_{i,j} = 0$ if $i \neq j$, and $\delta_{i,i} = 1$. It is easy to see that $\mathsf{B}[\mathbb{X}]$ does in fact have biproducts:

▶ **Lemma 31.** $\mathsf{B}[\mathbb{X}]$ *is a well-defined category with biproducts.*

If $\mathbb{X}$ is an additive symmetric monoidal category, then so is $\mathsf{B}[\mathbb{X}]$. The monoidal unit is the same as in $\mathbb{X}$, the tensor product of objects is:

$$(A_1, \ldots, A_n) \otimes (B_1, \ldots, B_m) = (A_1 \otimes B_1, \ldots, A_1 \otimes B_m, \ldots, A_n \otimes B_n)$$

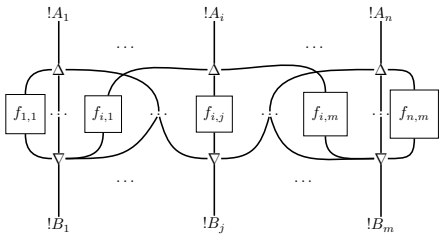
while the tensor product of maps is the standard Kronecker product of matrices.

▶ **Lemma 32.** *If* $\mathbb{X}$ *is an additive symmetric monoidal category, then so is* $\mathsf{B}[\mathbb{X}]$.

If $\mathbb{X}$ admits an additive bialgebra modality, then $\mathsf{B}[\mathbb{X}]$ is an additive monoidal storage category where the Seely isomorphisms are strict, i.e., equalities, and in particular is an additive linear category. We give the additive bialgebra modality of $\mathsf{B}[\mathbb{X}]$. The functor $! : \mathsf{B}[\mathbb{X}] \to \mathsf{B}[\mathbb{X}]$ is defined on objects as:

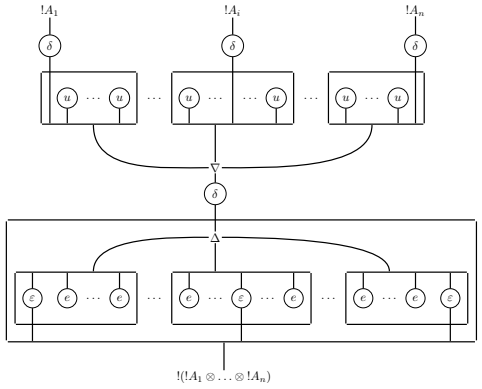
$$!(A_1, \ldots, A_n) = !A_1 \otimes \ldots \otimes !A_n$$

and on a map $[f_{i,j}] : (A_1, \ldots, A_n) \to (B_1, \ldots, B_m)$, $!([f_{i,j}])$ is represented in string diagrams below:



The bialgebra structure is given by the standard tensor product of bialgebras, the comonad

comultiplication $!(A_1, \ldots, A_n) \to !!(A_1, \ldots, A_n)$ is represented in string diagrams as:



while the comonad counit is the following matrix:

$$!A_1 \otimes \ldots \otimes !A_n \xrightarrow{\begin{bmatrix} \varepsilon_{A_1} \otimes e \otimes \ldots \otimes e \\ \ldots \\ e \otimes \ldots \otimes \varepsilon_{A_i} \otimes \ldots \otimes e \\ \ldots \\ e \otimes e \otimes \ldots \otimes \varepsilon_{A_n} \end{bmatrix}} (A_1, \ldots, A_n)$$

▶ **Lemma 33.** *If $\mathbb{X}$ has an additive bialgebra modality, then $\mathsf{B}[\mathbb{X}]$ is an additive monoidal storage category.*

Confluence of an Extension of Combinatory Logic by Boolean Constants*

Łukasz Czajka

DIKU, University of Copenhagen, Copenhagen, Denmark
luta@di.ku.dk

Abstract

We show confluence of a conditional term rewriting system CL-pc[1], which is an extension of Combinatory Logic by Boolean constants. This solves problem 15 from the RTA list of open problems. The proof has been fully formalized in the Coq proof assistant.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases combinatory logic, conditional linearization, unique normal form property, confluence

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.14

1 Introduction

Combinatory Logic is a term rewriting system defined by two rules:

$$\mathsf{K}xy \to x \qquad \mathsf{S}xyz \to xz(yz)$$

Using only S and K, it is possible to encode natural numbers via Church numerals. Any computable function may then be represented by a term in the system. However, a conditional C encoded in this way does not have a desirable property that $\mathsf{C}t_1t_2t_2 = t_2$ if t_1 encodes neither true nor false. It is therefore interesting to investigate extensions of Combinatory Logic incorporating a conditional directly. Perhaps the most natural such extension is CL-pc:

$$\begin{array}{llll} \mathsf{K}xy \to x & \mathsf{CT}xy \to x & \mathsf{C}zxx \to x \\ \mathsf{S}xyz \to xz(yz) & \mathsf{CF}xy \to y \end{array}$$

The system CL-pc is known to be not confluent [7]. One may thus try other ways of adding a conditional and Boolean constants to Combinatory Logic.

We show confluence of a conditional term rewriting system CL-pc[1] defined by the rules:

$$\begin{array}{llll} \mathsf{K}xy \to x & \mathsf{CT}xy \to x & \mathsf{C}zxy \to x \Leftarrow x = y \\ \mathsf{S}xyz \to xz(yz) & \mathsf{CF}xy \to y \end{array}$$

Confluence of this system[1] appears as problem 15 on the RTA list of open problems [5].

The equality in the side condition for the third rule for C in CL-pc[1] refers to equality in the system CL-pc[1] itself, thus the definition is circular. This circularity is an essential property of CL-pc[1] which distinguishes it from CL-pc.

* Supported by Marie Skłodowska-Curie action "InfTy", program H2020-MSCA-IF-2015, number 704111.

[1] Strictly speaking, in the literature the systems CL-pc, CL-pc[1] and CL-pc[L] also contain the rule $\mathsf{I}x \to x$. This rule could be added to our definitions without significantly changing the proofs. However, this would increase the number of cases to consider, making the proofs less readable. The formalization of our results uses the definitions from the literature.

A system related to CL-pc^1 is CL-pc$^{\mathrm{L}}$, which consists of all rules of CL-pc^1 plus:

$$\mathsf{C}zxy \to y \quad \Leftarrow \quad x = y$$

It is known that CL-pc$^{\mathrm{L}}$ is confluent [4]. However, the confluence proof in [4] essentially depends on a "semantic" argument to first establish $\mathsf{T} \neq_{\text{CL-pc}^{\mathrm{L}}} \mathsf{F}$. We provide a "syntactic" proof of confluence of both CL-pc^1 and CL-pc$^{\mathrm{L}}$.

The systems CL-pc^1 and CL-pc$^{\mathrm{L}}$ are conditional linearizations of CL-pc. The notion of conditional linearization was introduced in the hope of providing a simpler proof of Chew's theorem [2, 9] which states that all compatible term rewriting systems have the unique normal form (UN) property. Compatibility imposes certain restrictions on the term rewriting system, but it does not require termination or left-linearity. In particular, Chew's theorem is applicable to many term rewriting systems which are not confluent. For instance, CL-pc satisfies the conditions of Chew's theorem, but it is not confluent. As shown in [4], to prove the unique normal form property of a term rewriting system, it suffices to prove confluence of one of its conditional linearizations. The proof of Chew's theorem in [9] is quite complicated and uses a related but different approach, relying on left-right separated conditional linearizations instead of the more straightforward ones from [4]. The original proof by Chew [2] uses yet another different but related method, but Chew's proof was later found to contain a gap.

In general, the methods of the present paper are broadly related to the problem of establishing the UN property for classes of term rewriting systems which include non-left-linear non-confluent systems. Aside of Chew's theorem, some other work in this direction has been carried out in e.g. [8, 12, 13, 10, 6].

In order to increase confidence in the correctness of the main result of this paper, we have formalized our proof of confluence of CL-pc^1 in the Coq proof assistant. The formalization is available online[2]. It follows closely the development presented here. We used the CoqHammer [3] tool and the automated reasoning tactics included with it.

2 Proof overview

In this section we present an informal overview of the proof, trying to convey the underlying intuitions. Section 3 presents formal definitions of the notions informally motivated here, and Section 4 provides details of the proof itself.

We assume familiarity with basic term-rewriting [1, 11]. By $\to^*$ we denote the transitive-reflexive closure of a relation $\to$, by $\to^{\equiv}$ its reflexive closure, by $\leftrightarrow$ the symmetric closure, and by $=$ the reflexive-transitive-symmetric closure. We use $\equiv$ to denote identity of terms. By $\to^!$ we denote reduction to normal form, i.e., $t \to^! s$ if $t \to^* s$ and s is in normal form. By $\cdot$ we denote composition of relations, e.g. $t \to \cdot \leftarrow t'$ holds iff there exists t_0 such that $t \to t_0$ and $t_0 \leftarrow t'$. We use the standard notions of subterms and subterm occurrences, which could be formally defined by introducing the notion of positions. If t is a redex, i.e. $t \equiv \sigma l$ for some term l and substitution σ, then a subterm s *occurs below a variable position* of the redex t if s occurs in a subterm of t occurring at the position of a variable in l. The contraction in $t_1 \to t_2$ *occurs at the root* if t_1 is the contracted redex.

Let u be a normal form w.r.t. a relation $\to$. The relation $\to$ (or the underlying rewrite system) is *u-normal* if for every t such that $t = u$ we have $t \to^* u$.

[2] `http://www.mimuw.edu.pl/~lukaszcz/clc.tar.gz`

The most difficult part of our confluence proof is to show that CL-pc^1 is F-normal (Lemma 27). The confluence of CL-pc^1 (and CL-pc$^{\mathrm{L}}$) is then obtained by a relatively simple argument similar to the one used in [4] to derive the confluence of CL-pc$^{\mathrm{L}}$ from $\mathsf{T} \neq_{\text{CL-pc}} \mathsf{F}$.

An important observation is that $q_1 =_{\text{CL-pc}^1} q_2$ and $q_1 =_{\text{CL-pc}} q_2$ are in fact equivalent (Lemma 2). Hence, we will use $=_{\text{CL-pc}^1}$ and $=_{\text{CL-pc}}$ interchangeably. In particular, we actually prove that for any term q, if $q =_{\text{CL-pc}} \mathsf{F}$ then $q \to^*_{\text{CL-pc}^1} \mathsf{F}$.

A naive approach to prove this could be to proceed by induction on the length of the conversion $q =_{\text{CL-pc}} \mathsf{F}$. In the inductive step we would need to prove:

1. if $q \to^*_{\text{CL-pc}^1} \mathsf{F}$ and $q \to_{\text{CL-pc}} q'$ then $q' \to^*_{\text{CL-pc}^1} \mathsf{F}$,
2. if $q \to^*_{\text{CL-pc}^1} \mathsf{F}$ and $q \;{}_{\text{CL-pc}}\!\leftarrow q'$ then $q' \to^*_{\text{CL-pc}^1} \mathsf{F}$.

The second part is obvious, but the first one is hard. The difficulty stems from the existence of a non-trivial overlap between the rules for C. If $t_1 =_{\text{CL-pc}^1} t_2$ then $\mathsf{CF}t_1t_2 \to_{\text{CL-pc}^1} t_1$ by the third rule of CL-pc^1 and $\mathsf{CF}t_1t_2 \to_{\text{CL-pc}} t_2$ by the second rule of CL-pc. We do not know enough about t_1 and t_2 to easily infer that they have a common reduct in CL-pc^1.

One may try to strengthen the inductive hypothesis in the hope of making the first part easier to prove. A naive attempt would be to claim that *all* reductions starting from q end in F, instead of claiming that *some* reduction ends in F. This would make the first part trivial, but the second one would not go through as this is false in general, e.g., consider $\mathsf{KF}\Omega$ where $\Omega \equiv (\mathsf{SII})(\mathsf{SII})$ and $\mathsf{I} \equiv \mathsf{SKK}$.

The idea is to consider, for a given conversion $q =_{\text{CL-pc}} \mathsf{F}$, a certain set $\mathcal{S}(q =_{\text{CL-pc}} \mathsf{F})$ of reductions, all starting from q. The set $\mathcal{S}(q =_{\text{CL-pc}} \mathsf{F})$ depends on the exact form of $q =_{\text{CL-pc}} \mathsf{F}$. Then our two parts of the proof for the inductive step become:

1. if $\mathcal{S}(q =_{\text{CL-pc}} \mathsf{F})$ is nonempty and all reductions in it end in F, and $q \to_{\text{CL-pc}^1} q'$, then $\mathcal{S}(q' \;{}_{\text{CL-pc}}\!\leftarrow q =_{\text{CL-pc}} \mathsf{F})$ is nonempty and all reductions in it end in F,
2. if $\mathcal{S}(q =_{\text{CL-pc}} \mathsf{F})$ is nonempty and all reductions in it end in F, and $q \;{}_{\text{CL-pc}}\!\leftarrow q'$, then $\mathcal{S}(q' \to_{\text{CL-pc}} q =_{\text{CL-pc}} \mathsf{F})$ is nonempty and all reductions in it end in F.

The hope is that if we define $\mathcal{S}(q =_{\text{CL-pc}} \mathsf{F})$ appropriately, then showing both parts will become feasible.

Essentially, the set $\mathcal{S}(q =_{\text{CL-pc}} \mathsf{F})$ will be encoded in the labeling of certain constants in q. The labels determine which contractions are permitted when a given constant appears as the leftmost constant in a redex[3]. At present the author does not know an "explicit" characterization of the set of reductions $\mathcal{S}(q =_{\text{CL-pc}} \mathsf{F})$ implicitly defined by the labelings described below.

Terms with the leftmost constant labeled will be called "significant", or s-terms, whereas others will not contain any labels and will be called "insignificant", or i-terms (cf. Definition 3). Reductions occurring in i-terms will be "insignificant", or i-reductions. A "significant" contraction, or s-contraction, will be a contraction of a term with the leftmost constant labeled, in a way permitted by the label of the leftmost constant. Contraction of a redex in which the leftmost constant is not labeled is not permitted in s-contractions. See Definition 4. The intuition is that we do not need to care about the expansions and contractions occurring in "insignificant" subterms of a given term, since they cannot influence the s-reductions starting from this term and ending in F.

The set $\mathcal{S}(q =_{\text{CL-pc}} \mathsf{F})$ will be encoded in a labeled variant[4] t of q, and it will consist of all s-reductions starting from t and ending in a normal form (w.r.t. s-contraction). Strictly

[3] E.g. in the redex $\mathsf{CT}t_1t_2$ the constant C is the leftmost constant.

[4] By a "labeled variant" of a term q we mean a term with certain constants labeled which is identical with q when the labels are "erased".

speaking, we have just silently shifted from considering contractions in "plain" terms of the system CL-pc[1] to contractions in their labeled variants, in a different rewriting system which we have not yet defined. In particular, we will actually be interested in s-reductions ending in a labeled variant F_1 of F. However, it will be later shown that s-reductions defined on labeled terms may be "erased" to appropriate reductions in the system CL-pc[1]. In the next section we define the system CL-pc^s (Definition 4) over labeled terms (Definition 3) which will give precise rules of s-contraction. In this section we only give informal motivations.

The labels constrain the ways in which s-redexes may be contracted and encode permissible s-reductions to F_1. Each term decomposes into a "significant" prefix and an "insignificant" suffix (cf. 1 in Definition 8). The "significant" prefix contains all labeled constants and no unlabeled constants. The "insignificant" suffix consists of all "insignificant" subterms. All constants in the "insignificant" suffix are unlabeled. This is analogous to the existence of a needed prefix and a non-needed suffix in orthogonal TRSs [11, Section 9.2.2]. An "insignificant" subterm does not overlap with any needed redexes. In particular, it does not contain any needed redexes. No position inside an "insignificant" subterm (dynamically) traces to F_1 along any s-reduction to F_1 (cf. [11, Definition 8.6.7]). In contrast, each s-redex needs to be either s-contracted or erased by a rule for C_2 (see Definition 4) in any s-reduction to F_1. Each position of a labeled constant either traces to F_1 along a given s-reduction to F_1, or is erased in that s-reduction by a rule for C_2. An s-reduct of an s-term is always also an s-term (cf. 5 in Definition 8).

We write $t \to_s t'$ for one-step reduction in CL-pc^s. We use the abbreviation s-NF for CL-pc^s-normal form. We write $t \Downarrow_{\mathsf{F}_1}$ when, among other conditions to be defined later, t is complete, i.e. terminating and confluent, w.r.t. s-reductions with F_1 as the normal form (cf. Definition 8).

With the set $\mathcal{S}(q =_{\text{CL-pc}} \mathsf{F})$ coded by labels, the two parts of the inductive step become:

1. if t is a labeled variant of q such that $t \Downarrow_{\mathsf{F}_1}$, and $q \to_{\text{CL-pc}} q'$, then there exists a labeled variant t' of q' such that $t' \Downarrow_{\mathsf{F}_1}$ (cf. Corollary 15),
2. if t is a labeled variant of q such that $t \Downarrow_{\mathsf{F}_1}$, and $q \;{}_{\text{CL-pc}}\!\leftarrow q'$, then there exists a labeled variant t' of q' such that $t' \Downarrow_{\mathsf{F}_1}$ (cf. Corollary 25).

Now we provide some explanations on how the terms will be labeled. For this purpose we analyze why the second part fails when we take $\mathcal{S}(q =_{\text{CL-pc}} F)$ to be the set of all reductions starting from q. We indicate how to introduce the labeled variants so as to make the second part go through while still retaining the feasibility of showing the first part.

Suppose $q \;{}_{\text{CL-pc}}\!\leftarrow q'$ at the root and we have already decided on the labeled variant t of q. We need to decide on a labeled variant t' of q', and assign appropriate meaning to the labels, in such a way that the second part goes through. In short, in t' we preserve the labelings of the subterms of q' which are copied to q in $q' \to_{\text{CL-pc}} q$, we do not label the new subterms of q' which are erased in $q' \to_{\text{CL-pc}} q$ (they become i-terms), and we ensure that i-terms and cannot influence *any* s-reduction from t' to F_1. First of all, if t is an i-term, i.e., $t \equiv q$, then we may take $t' \equiv q'$. So assume t is an s-term. Then there are the following possibilities.

- If $q' \equiv \mathsf{C}\mathsf{T}qq_0 \to_{\text{CL-pc}} q$ then q_0 is a new subterm. We take $t' \equiv \mathsf{C}_1\mathsf{T}_1tq_0$. The labeling C_1 of C will be interpreted as not permitting contraction by the third rule, i.e., in CL-pc^s we will only have the rules $\mathsf{C}_1\mathsf{T}_1xy \to x$ and $\mathsf{C}_1\mathsf{F}_1xy \to y$. This ensures that q_0 gets erased in every s-reduction of t' to F_1.
- The case when $q' \equiv \mathsf{C}\mathsf{F}q_0q \to_{\text{CL-pc}} q$ is analogous: we take $t' \equiv \mathsf{C}_1\mathsf{F}_1q_0t$.
- If $q' \equiv \mathsf{C}q_0qq \to_{\text{CL-pc}} q$ by the third rule, then q_0 is a new term. We take $t' \equiv \mathsf{C}_2q_0tt$. In

the system CL-pc^s we have two rules for C_2

$$\begin{array}{lcll} \mathsf{C}_2 zxy & \to & x & \Leftarrow \quad |x| =_{\text{CL-pc}^1} |y| \\ \mathsf{C}_2 zxy & \to & y & \Leftarrow \quad |x| =_{\text{CL-pc}^1} |y| \end{array}$$

where $|x| =_{\text{CL-pc}^1} |y|$ means that the "erasures" of the labeled terms substituted for x and y must be equal in CL-pc^1 for the rule to be applicable. These rules ensure that q_0 cannot influence any s-reduction of t' to F_1 – it gets erased in each.

The presence of the second rule for CL-pc^s is not a problem, because we will only consider terms terminating in CL-pc^s. Whenever the second rule is applicable, so is the first one, hence if all maximal s-reductions end in F_1, then there is an s-reduction ending in F_1 which does not use the second rule for C_2 (Lemma 26). It will be easy to "erase" an s-reduction not using the second rule for C_2 to obtain a reduction in CL-pc^1 (Lemma 7).

- If $q' \equiv \mathsf{K} q q_0 \to_{\text{CL-pc}^1} q$ then we take $t' \equiv \mathsf{K}_1 t q_0$. The rule for K_1 in CL-pc^s is $\mathsf{K}_1 xy \to x$.
- If $q' \equiv \mathsf{S} q_1 q_2 q_3 \to_{\text{CL-pc}^1} q_1 q_3 (q_2 q_3) \equiv q$ then we run into a problem with our labeling approach, because the labeled variants of the distinct occurrences of q_3 may be distinct. Suppose t_1 is the labeled variant of q_1, the term t_2 of q_2, the term t_3 of the first q_3, and t_3' of the second q_3. We cannot just arbitrarily choose e.g. t_3 and say that $\mathsf{S}_1 t_1 t_2 t_3$ is the labeled variant of q', because contracting $\mathsf{S}_1 t_1 t_2 t_3$ yields $t_1 t_3 (t_2 t_3)$, not $t_1 t_3 (t_2 t_3')$, and now the second occurrence of q_3 has the wrong labeling.

  A solution is to remember both labeled variants of q_3. So the labeled variant of q' would be e.g. $\mathsf{S}_1 t_1 t_2 \langle t_3, t_3' \rangle$. In CL-pc^s the rule for S_1 would be

  $$\mathsf{S}_1 x_1 x_2 \langle x_3, x_3' \rangle \to x_1 x_3 (x_2 x_3').$$

  However, once we introduce such pairs, terms of the form $\mathsf{S}_1 t_1 t_2 \langle t_3, t_3' \rangle$ may appear in the terms being expanded. This is not a problem for any of the rules of CL-pc^1 except the rule for S, because the right sides of all other rules are variables.

  Consider for instance $q \equiv q_0 q_3 (\mathsf{S} q_1 q_2 q_3) \;{}_{\text{CL-pc}^1}\!\leftarrow \mathsf{S} q_0 (\mathsf{S} q_1 q_2) q_3 \equiv q'$. Suppose $t \equiv t_0 t_3 (\mathsf{S}_1 t_1 t_2 \langle t_3, t_3' \rangle)$. Now the term q_3 has three possibly distinct labeled variants, and we need to remember all of them in a tuple. We will thus introduce a new labeling of S for every possible labeling of the right side $xz(yz)$ of the rule for S in the system CL-pc^1.

  By introducing the tuples in the labelings we in essence put constraints on the order in which s-redexes may be contracted (think of all reductions inside a tuple as "really" occuring after the surrounding S-redex is contracted). At present the author does not know a precise "explicit" characterization of these constraints.

Note that by labeling C differently in $\mathsf{C}\mathsf{F} q_1 q_2$ and $\mathsf{C} q_0 q q$ we effectively eliminated in CL-pc^s the problematic non-trivial overlap occuring in CL-pc^1. Now a new "insignificant" term created in an expansion cannot later on appear in place of a "significant" term as a result of an "incompatible" contraction. A redex inside an "insignificant" subterm cannot suddenly become needed in an s-reduction – it is erased in any s-reduction to normal form.

We also need to ensure that we can handle the first part of the inductive step when $q \to_{\text{CL-pc}} q'$. Suppose t is the labeled variant of q. We need to find a labeled variant for q'. For simplicity assume that there is only one position in t which corresponds to the position of the contraction in q. If the contraction occurs inside an i-term in t, then it does not matter and we may label q' in the same way as q. If an s-term is contracted in a way permitted for significant contraction, then it is also obvious how to label q' – just take the labeled variant of q' to be the reduct of the labeled variant of q. But what if neither of the two holds?

For instance, what if $t \equiv \mathsf{C}_1 t_0 t_1 t_2$ but $q \equiv \mathsf{C} q_0 q' q' \to_{\text{CL-pc}^1} q'$? This possibility is not problematic, provided that $t_0 \to_s^* \mathsf{T}_1$ or $t_0 \to_s^* \mathsf{F}_1$, which will be the case because t_0 was

"obtained" from T_1 or F_1 by a conversion with the intermediate terms labeled appropriately (cf. 2 in Definition 8 and 6 in Lemma 9). If e.g. $t_0 \to_s^* \mathsf{T}_1$ then we take t_1 to be the labeling of q'. We then have $\mathsf{C}_1 t_0 t_1 t_2 \to_s^* \mathsf{C}_1 \mathsf{T}_1 t_1 t_2$ and the contraction $\mathsf{C}_1 \mathsf{T}_1 t_1 t_2 \to_s t_1$ is permitted for "significant" contractions.

The last problematic case is when e.g. $t \equiv \mathsf{C}_2 \mathsf{F} t_1 t_2$ is the labeling of $q \equiv \mathsf{C}\mathsf{F} q_1 q'$, and $q \to_{\text{CL-pc}} q'$ by the second rule. However, because $\mathsf{C}_2 \mathsf{F} t_1 t_2$ was "obtained" from $\mathsf{C}_2 q t' t'$ we will have $|t_1| =_{\text{CL-pc}^1} |t_2|$ (cf. 3 in Definition 8). Then the second rule for C_2 in CL-pc^s is applicable and we may take t_2 as the labeling of q'.

3 Definitions

This section is devoted to fixing notation and introducing definitions of various technical concepts. First, we clarify the formal definition of conditional term rewriting systems. For more background on conditional rewriting see e.g. [11].

▶ **Definition 1.** A *conditional rewrite rule* is a rule of the form $l \to r \Leftarrow P(x_1, \ldots, x_n)$, where l is not a variable, $\mathrm{Var}(r) \subseteq \mathrm{Var}(l)$, $x_1, \ldots, x_n \in \mathrm{Var}(l)$, and $P(x_1, \ldots, x_n)$ is the *condition* of the rule, with P a fixed predicate on terms. The predicate P may refer to the conversion relation $=$ of the conditional term rewriting system being defined. A term t is a *redex* (*contractum*) by this rule if there is a substitution σ such that $t \equiv \sigma l$ ($t \equiv \sigma r$) and $P(\sigma(x_1), \ldots, \sigma(x_n))$ holds. A *conditional term rewriting system* R is a set of conditional rewrite rules. Because the conditions in the rules may refer to the conversion relation of R, the definition is circular. Formally, an R-contraction $q \to_R q'$ is defined in the following way. Define R_0 to be the system R but using the equality relation in place of $=$ in the conditions, and R_{n+1} to be the system R with the conversion relation $=_{R_n}$ of R_n used in place of $=$. We then define $q \to_R q'$ to hold if there is $n \in \mathbb{N}$ with $q \to_{R_n} q'$. The least such n is called the *level* of the contraction. If the conditions are continuous w.r.t. $=$ then the relation $\to_R$ is a fixpoint of the above construction, i.e., it is the contraction relation of the system R_∞ which uses $=_R$ in place of $=$. Let $\sim$ be a binary relation on terms. If for any substitution σ such that $P(\sigma(x_1), \ldots, \sigma(x_n))$ holds, and any σ' such that $\sigma(x) \sim \sigma'(x)$ for all variables x, also $P(\sigma'(x_1), \ldots, \sigma(x_1'))$ holds, then the condition $P(x_1, \ldots, x_n)$ is *stable under* $\sim$.

The following is a simple but crucial observation, which implies that it suffices to consider conversions in CL-pc. A generalization of this fact was already shown in [4, Lemma 3.7]. The proof is by induction on the maximum level of the contractions/expansions in $q =_{\text{CL-pc}^{\text{L}}} q'$.

▶ **Lemma 2.** *The following are equivalent:* $q =_{\text{CL-pc}} q'$, $q =_{\text{CL-pc}^1} q'$, *and* $q =_{\text{CL-pc}^{\text{L}}} q'$.

▶ **Definition 3.** We define *insignificant terms*, or *i-terms*, to be the terms of CL-pc^1, i.e., terms over the signature $\Sigma = \{@, \mathsf{C}, \mathsf{T}, \mathsf{F}, \mathsf{K}, \mathsf{S}\}$ where $@$ is a binary function symbol and the other symbols are constants. We write $t_1 t_2$ instead of $@(t_1, t_2)$. The set of *labeled terms*, or *l-terms*, is the set of terms over the signature consisting of the symbols of Σ, the *labeled constants* $\mathsf{C}_1, \mathsf{C}_2, \mathsf{T}_1, \mathsf{F}_1, \mathsf{K}_1$ and $\mathsf{S}^{n_0,\ldots,n_k}$ for each $k, n_1, \ldots, n_k \in \mathbb{N}_+$, and an n-ary function symbol P^n for each $n \in \mathbb{N}_+$. We write $\langle t_1, \ldots, t_n \rangle$ instead of $\mathsf{P}^n(t_1, \ldots, t_n)$. We adopt the convention $\langle t \rangle \equiv t$. If $t \equiv \langle t_1, \ldots, t_n \rangle$ with $n > 1$, then we say that t is a *tuple of length* n. Note that $\langle t \rangle \equiv t$ is just a notational convention. We say that $\langle t_1, \ldots, t_n \rangle$ is a tuple *only when* $n > 1$.

An *erasure* of an l-term is defined as follows:

- an i-term is an erasure of itself,
- C is an erasure of C_1 and C_2; T is an erasure of T_1; F is an erasure of F_1; K is an erasure of K_1; S is an erasure of $\mathsf{S}^{n_1,\ldots,n_k}$,

- if q_1, q_2 are erasures of t_1, t_2, respectively, then q_1q_2 is an erasure of t_1t_2,
- if q_i is an erasure of t_i, for some $1 \leq i \leq n$, then q_i is an erasure of $\langle t_1, \ldots, t_n\rangle$.

The *leftmost erasure* of t, denoted $|t|$, is the erasure in which we always choose $i = 1$ in the last point above. We write $t \succ q$ if *every* erasure of t is identical with q.

We define *significant terms*, or *s-terms*, inductively.

- Any labeled constant is an s-term.
- If t_1 is an s-term and t_2 is an l-term, then t_1t_2 is an s-term.

In other words, an s-term is an l-term whose leftmost constant is labeled.

In what follows t, t_1, t_2, r, r_1, r_2, s, s_1, etc. stand for l-terms; and q, q_1, q_2, etc. stand for i-terms; unless otherwise qualified. Also, whenever we talk about terms without further qualification, we implicitly assume them to be l-terms.

▶ **Definition 4.** The system CL-pc^s is defined by the following *significant reduction rules*:

$$\begin{array}{rcl@{\qquad}rcll} \mathsf{C}_1\mathsf{T}_1xy & \to & x & \mathsf{C}_2zxy & \to & x & \Leftarrow \quad |x| =_{\text{CL-pc}^1} |y| \\ \mathsf{C}_1\mathsf{F}_1xy & \to & y & \mathsf{C}_2zxy & \to & y & \Leftarrow \quad |x| =_{\text{CL-pc}^1} |y| \\ \mathsf{K}_1xy & \to & x & & & & \end{array}$$

$$\mathsf{S}^{\vec{n}}x\langle y_1, \ldots, y_k\rangle\langle \vec{z}_0, \ldots, \vec{z}_k\rangle \quad \to \quad x\langle \vec{z}_0\rangle\langle (y_1\langle \vec{z}_1\rangle), \ldots, (y_k\langle \vec{z}_k\rangle)\rangle \quad \Leftarrow \quad \varphi$$

where

$$\begin{aligned} \varphi \quad \equiv \quad & |z_{i,j}| =_{\text{CL-pc}^1} |z_{i',j'}| \text{ for } i, i' = 0, \ldots, k,\ j = 1, \ldots, n_i,\ j' = 1, \ldots, n_{i'}, \text{ and} \\ & |y_i| =_{\text{CL-pc}^1} |y_j| \text{ for } i, j = 1, \ldots, k, \end{aligned}$$

and $\vec{n}$ stands for $n_0, \ldots, n_k$, and $\vec{z}_i$ stands for $z_{i,1}, \ldots, z_{i,n_i}$, for $i = 0, \ldots, k$. When dealing with terms whose leftmost constant is $\mathsf{S}^{n_0,\ldots,n_k}$, we will often use this kind of vector notation. Recall the convention $\langle t\rangle \equiv t$. Hence, if e.g. $n_0 = 1$, then $\langle \vec{z}_0\rangle \equiv \langle z_{0,1}\rangle \equiv z_{0,1}$ in the above rule. The condition φ ensures that the leftmost erasures of all $z_{i,j}$ are convertible in CL-pc^1, and that the leftmost erasures of all y_i are convertible in CL-pc^1. Some examples of significant reduction rules for $\mathsf{S}^{\vec{n}}$ (omitting the conditions) are:

$$\begin{array}{rcl} \mathsf{S}^{1,1}xy_1\langle z_{0,1}, z_{1,1}\rangle & \to & xz_{0,1}(y_1z_{1,1}) \\ \mathsf{S}^{1,2,1}x\langle y_1, y_2\rangle\langle z_{0,1}, z_{1,1}, z_{1,2}, z_{2,1}\rangle & \to & xz_{0,1}\langle y_1\langle z_{1,1}, z_{1,2}\rangle, y_2z_{1,2}\rangle \\ \mathsf{S}^{2,2}xy_1\langle z_{0,1}, z_{0,2}, z_{1,1}, z_{1,2}\rangle & \to & x\langle z_{0,1}, z_{0,2}\rangle(y_1\langle z_{1,1}, z_{1,2}\rangle) \end{array}$$

For instance, the condition for the second of these rules states that $|y_1| =_{\text{CL-pc}^1} |y_2|$, $|z_{0,1}| =_{\text{CL-pc}^1} |z_{1,1}|$, $|z_{0,1}| =_{\text{CL-pc}^1} |z_{1,2}|$, $|z_{0,1}| =_{\text{CL-pc}^1} |z_{2,1}|$, $|z_{1,1}| =_{\text{CL-pc}^1} |z_{1,2}|$, etc.

Note that the equality $=_{\text{CL-pc}^1}$ in the conditions refers to the system CL-pc^1, not CL-pc^s. Note also that all rules of CL-pc^s are linear, disregarding the side-conditions.

Reduction by a rule in CL-pc^s is called *significant reduction*, or *s-reduction*. One-step s-reduction is denoted by $\to_s$. Analogously, we use the terminology and notation of s-contraction, s-expansion, s-redex, s-normal form (s-NF), etc. Note that every s-redex is an s-term. We write $t \to_{s-} t'$ if $t \to_s t'$ and the s-contraction is not by the second rule for C_2 and it does not occur inside a tuple.

An *i-redex* is a CL-pc^1-redex which is also an i-term. An l-term t_1 is said to *i-reduce* to t_2, denoted $t_1 \to_i t_2$, if $t_1 \to_{\text{CL-pc}^1} t_2$ and the redex contracted in t_1 is an i-term. An l-term t_1 is said to *i-expand* to t_2 if $t_2 \to_i t_1$. We write $t_1 \to_{i,s} t_2$ if $t_1 \to_i t_2$ or $t_1 \to_s t_2$.

Actually, we will consider mostly l-terms whose all erasures are identical. For such a term an s-contraction by a rule for $\mathsf{S}^{\vec{n}}$ in CL-pc^s naturally corresponds to a CL-pc^1-contraction on its erasure. We could get rid of the side conditions in the rules for $\mathsf{S}^{\vec{n}}$ and

consider exclusively terms whose all erasures are identical. But then we would need to require i/s-contractions/expansions to always occur "in the same way" (modulo labeling) in all components of a tuple. This would complicate the inductive proofs concerning the relations $\to_s$, $\to_i$, etc. Hence, the role of the conditions in the rules for $\mathsf{S}^{\vec{n}}$ is purely technical.

▶ **Lemma 5.** *The system* CL-pc^s *is terminating.*

Proof. The number of labeled constants decreases with each s-contraction. ◀

▶ **Lemma 6.** *If* $t_1 \to_s t_2$ *then* $|t_1| =_{\text{CL-pc}^1} |t_2|$.

The above simple lemma implies that the conditions in significant reduction rules are stable under s-reduction and s-expansion. It is obvious that they are also stable under i-reduction and i-expansion.

▶ **Lemma 7.** *If* $t \succ q$ *and* $t \to_{s-} t'$ *then there is* q' *with* $q \to_{\text{CL-pc}^1} q'$ *and* $t' \succ q'$.

Proof. Because all erasures of t are identical and the second rule for C_2 is not used, the $s-$-reduction may be simulated by a CL-pc^1-reduction in an obvious way. Because the $s-$-contraction does not occur inside a tuple, all erasures of t' are still identical. ◀

In the next definition we introduce the predicate $\Downarrow_{\mathsf{F}_1}$ and the notion of standard l-terms. Intuitively, an l-term t is standard if the labelings in t have the meaning we intend to assign them, i.e. if t is a term obtained by the process informally described in the previous section.

▶ **Definition 8.** An l-term t is *standard* if for every subterm t' of t the following hold:

1. t' is either an i-term, an s-term or a tuple,
2. if $t' \equiv \mathsf{C}_1 t_0 t_1 t_2$ and t_0 is in s-NF, then $t_0 \equiv \mathsf{T}_1$ or $t_0 \equiv \mathsf{F}_1$,
3. if $t' \equiv \mathsf{C}_2 t_0 t_1 t_2$ then $|t_1| =_{\text{CL-pc}^1} |t_2|$,
4. if $t' \equiv \mathsf{S}^{n_0,\dots,n_k} t_0 t_1 t_2$ then t_2 is a tuple of length $\sum_{i=0}^{k} n_k$ and if $k > 1$ then t_1 is a tuple of length k,
5. if t' is an s-term and $t' \to_s^* t''$, then t'' is also an s-term,
6. if $t' \equiv \langle t_1, \dots, t_n \rangle$ with $n > 1$, then none of $t_1, \dots, t_n$ is a tuple.

An l-term t is *strongly standard* if $t \to_s^* t'$ implies that t' is standard. We write $t \Downarrow_{\mathsf{F}_1}$ if t is strongly standard and has no s-NFs other than F_1, i.e. if $t \to_s^! t'$ then $t' \equiv \mathsf{F}_1$.

Point 1 in Definition 8 essentially ensures that a standard term may be decomposed into a "significant" prefix and an "insignificant" suffix. A labeled term which is not standard is e.g. CT_1, because it is neither an s-term, nor an i-term, nor a tuple. Other examples of non-standard terms are: $\mathsf{C}_1\mathsf{TFF}$, $\mathsf{C}_2\mathsf{CTF}$, $\mathsf{S}^{1,1,1}\mathsf{TTT}$, $\mathsf{K}_1\mathsf{FF}$, $\mathsf{C}_2\mathsf{CT}_1\mathsf{T}$, $\mathsf{S}^{1,1}\mathsf{CC}\langle\mathsf{T}_1,\mathsf{T}_1\rangle$, $\mathsf{K}_1\langle\mathsf{T}_1,\mathsf{T}_1\rangle\mathsf{T}_1$, $\langle\langle\mathsf{C},\mathsf{C}\rangle,\mathsf{C}\rangle$. Examples of standard terms which are not strongly standard are: $\mathsf{S}^{1,1}\mathsf{C}_1\mathsf{C}\langle\mathsf{T}_1,\mathsf{T}_1\rangle$, $\mathsf{S}^{1,1}\mathsf{C}_1\mathsf{C}\langle\mathsf{T},\mathsf{T}\rangle\mathsf{T}$, $(\mathsf{K}_1\mathsf{C}_1\mathsf{T})\mathsf{TTT}$.

▶ **Lemma 9.**

1. *Any i-term is standard.*
2. *Any labeled constant is standard.*
3. *Every subterm of a standard term is also standard.*
4. *Every subterm of a term to which some strongly standard term s-reduces, is strongly standard.*
5. *If $t_1 t_2$ is standard then t_1 is not a tuple.*
6. *If $\mathsf{C}_1 t_0 t_1 t_2$ is a subterm of a strongly standard term, then $t_0 \to_s^* \mathsf{T}_1$ or $t_0 \to_s^* \mathsf{F}_1$.*

Proof. Follows from definitions. For the last point one also needs Lemma 5. ◀

4 Confluence proof

We now give technical details of our confluence proof. As outlined in Section 2, we show:
1. if $t \succ q$ and $t \Downarrow_{\mathsf{F}_1}$, and $q \to_{\text{CL-pc}} q'$, then there is t' with $t' \succ q'$ and $t' \Downarrow_{\mathsf{F}_1}$ (Corollary 15),
2. if $t \succ q$ and $t \Downarrow_{\mathsf{F}_1}$, and $q \;{}_{\text{CL-pc}}\!\leftarrow q'$, then there is t' with $t' \succ q'$ and $t' \Downarrow_{\mathsf{F}_1}$ (Corollary 25).

The first part is proven by showing that CL-pc-reductions in q may be simulated by i-reductions and s-reductions in t, and that i/s-reductions preserve $\Downarrow_{\mathsf{F}_1}$ (Lemma 13 and Lemma 14). For the second part, we show that CL-pc-expansions in q may be simulated by i-expansions and a-expansions (Definition 16) in t. The technical notion of a-expansion is needed to ensure that the new subterms of t' are labeled appropriately, in the way outlined in Section 2 (s-contraction by itself does not put any labeling restrictions on the terms erased in the contraction). Moreover, a-expansion is also needed to facilitate the proof that $t' \Downarrow_{\mathsf{F}_1}$ (see the discussion before Definition 16). Plain s-expansion does not necessarily preserve $\Downarrow_{\mathsf{F}_1}$, while a-expansion does (Lemma 24).

In other words, we show that CL-pc-reductions (expansions) in unlabeled terms may be simluated by i/s-reductions (i/a-expansions) in their labeled variants, and that i/s-reductions (i/a-expansions) preserve $\Downarrow_{\mathsf{F}_1}$. A conversion $q =_{\text{CL-pc}} \mathsf{F}$ can then be translated into a conversion $t =_{i,s,a} \mathsf{F}_1$ with no s-expansions or a-reductions, and with $t \succ q$. For instance, a conversion in CL-pc

$$\begin{aligned}&\mathsf{F} \leftarrow \mathsf{C}(\mathsf{KF}\Omega)\mathsf{FF} \leftarrow \mathsf{C}(\mathsf{KF}\Omega)\mathsf{F}(\mathsf{CTF}(\mathsf{KF}\Omega)) \rightarrow \mathsf{CFF}(\mathsf{CTF}(\mathsf{KF}\Omega)) \leftarrow\\ &\quad \mathsf{CFF}(\mathsf{C}(\mathsf{KTF})\mathsf{F}(\mathsf{KF}\Omega)) \rightarrow \mathsf{C}(\mathsf{KTF})\mathsf{F}(\mathsf{KF}\Omega) \rightarrow \mathsf{C}(\mathsf{KTF})\mathsf{FF} \rightarrow \mathsf{F} \leftarrow\\ &\quad \mathsf{KF}(\mathsf{CF}) \leftarrow \mathsf{SKCF} \leftarrow \mathsf{SKC}(\mathsf{KF}\Omega) \rightarrow \mathsf{SKCF}\end{aligned}$$

will be translated to

$$\begin{aligned}&\mathsf{F}_1 \;{}_a\!\leftarrow \mathsf{C}_2(\mathsf{KF}\Omega)\mathsf{F}_1\mathsf{F}_1 \;{}_a\!\leftarrow \mathsf{C}_2(\mathsf{KF}\Omega)\mathsf{F}_1(\mathsf{C}_1\mathsf{T}_1\mathsf{F}_1(\mathsf{KF}\Omega)) \rightarrow_i \mathsf{C}_2\mathsf{FF}_1(\mathsf{C}_1\mathsf{T}_1\mathsf{F}_1(\mathsf{KF}\Omega)) \;{}_a\!\leftarrow\\ &\quad \mathsf{C}_2\mathsf{FF}_1(\mathsf{C}_1(\mathsf{K}_1\mathsf{T}_1\mathsf{F})\mathsf{F}_1(\mathsf{KF}\Omega)) \rightarrow_s \mathsf{C}_1(\mathsf{K}_1\mathsf{T}_1\mathsf{F})\mathsf{F}_1(\mathsf{KF}\Omega) \rightarrow_i \mathsf{C}_1(\mathsf{K}_1\mathsf{T}_1\mathsf{F})\mathsf{F}_1\mathsf{F} \rightarrow_s^* \mathsf{F}_1 \;{}_a\!\leftarrow\\ &\quad \mathsf{K}_1\mathsf{F}_1(\mathsf{CF}) \;{}_a\!\leftarrow \mathsf{S}^{1,1}\mathsf{K}_1\mathsf{C}\langle\mathsf{F}_1,\mathsf{F}\rangle \;{}_{a,i}^{*}\!\leftarrow \mathsf{S}^{1,1}\mathsf{K}_1\mathsf{C}\langle\mathsf{K}_1\mathsf{F}_1\Omega,\mathsf{KF}\Omega\rangle \rightarrow_{s,i}^* \mathsf{S}^{1,1}\mathsf{K}_1\mathsf{C}\langle\mathsf{F}_1,\mathsf{F}\rangle\end{aligned}$$

Since $\mathsf{F}_1 \Downarrow_{\mathsf{F}_1}$ and we prove that i/s-reductions and i/a-expansions preserve $\Downarrow_{\mathsf{F}_1}$, we may conclude that $t \Downarrow_{\mathsf{F}_1}$. Then by the definition of $\Downarrow_{\mathsf{F}_1}$ we obtain a significant reduction $t \to_s^* \mathsf{F}_1$. In fact, the reduction may be assumed to be a $s-$-reduction (Lemma 26). By Lemma 7 this reduction $t \to_{s-}^* \mathsf{F}_1$ may be translated into a CL-pc^1-reduction by erasing the labelings. Hence finally $q \to_{\text{CL-pc}^1}^* \mathsf{F}$ (Lemma 27).

We first show that a CL-pc-contraction may be simulated by i-reductions and s-reductions.

▶ **Lemma 10.** *If t is strongly standard, $t \succ q$ and $q \to_{\text{CL-pc}} q'$, then there exists a term t' such that $t \to_{i,s}^* t'$ and $t' \succ q'$.*

Proof. Induction on the size of t. First assume t is not a tuple and q is the CL-pc-redex contracted in $q \to_{\text{CL-pc}} q'$. If $t \equiv q$ then $t \equiv q \to_i q'$ and we may take $t' \equiv q'$. If $t \not\equiv q$ then t is not an i-term because $t \succ q$. Hence by 1 in Definition 8 we conclude that t is an s-term. We have the following possibilities.

- If $q \equiv \mathsf{CT}q_1q_2 \to_{\text{CL-pc}} q_1 \equiv q'$ then the leftmost constant in t is either C_1 or C_2.
  - If $t \equiv \mathsf{C}_1\mathsf{T}_1t_1t_2$ then $t \to_s t_1$ and $t_1 \succ q_1$, so we may take $t' \equiv t_1$.
  - The case $t \equiv \mathsf{C}_1\mathsf{T}t_1t_2$ is impossible by 2 in Definition 8.
  - If $t \equiv \mathsf{C}_2t_0t_1t_2$ then $t_1 \succ q_1$ and $|t_1| =_{\text{CL-pc}^1} |t_2|$ by 3 in Definition 8. Thus $t \to_s t_1$ and we may take $t' \equiv t_1$.
- If $q \equiv \mathsf{CF}q_1q_2 \to_{\text{CL-pc}} q_2$ then the argument is analogous. Note that the presence of the second rule for C_2 is necessary here.

- If $q \equiv \mathsf{C}q_0q_1q_1 \to_{\text{CL-pc}} q_1$ then $t \equiv \mathsf{C}'t_0t_1t_2$ with $\mathsf{C}' \in \{\mathsf{C}_1, \mathsf{C}_2\}$, $t_0 \succ q_0$, $t_1 \succ q_1$ and $t_2 \succ q_1$.
  - If $\mathsf{C}' \equiv \mathsf{C}_1$ then $t_0 \to_s^* \mathsf{T}_1$ or $t_0 \to_s^* \mathsf{F}_1$ by Lemma 9. Hence $t \to_s^* t_1$ or $t \to_s^* t_2$. In the first case we may take $t' \equiv t_1$, and in the second we take $t' \equiv t_2$.
  - If $\mathsf{C}' \equiv \mathsf{C}_2$ then $t \to_s t_1$ because $|t_1| \equiv |t_2| \equiv q_1$. Thus we take $t' \equiv t_1$.
- If $q \equiv \mathsf{K}q_1q_2 \to_{\text{CL-pc}} q_1$ then $t \equiv \mathsf{K}_1t_1t_2 \to_s t_1$ with $t_1 \succ q_1$. We take $t' \equiv t_1$.
- If $q \equiv \mathsf{S}q_0q_1q_2 \to_{\text{CL-pc}} q_0q_2(q_1q_2)$ then $t \equiv \mathsf{S}^{\vec{n}}s\langle t_1, \dots, t_k\rangle\langle \vec{r}_0, \dots, \vec{r}_k\rangle$ where the conventions regarding the vector notation are as in Definition 4, and $s \succ q_0$, and $t_i \succ q_1$ for $i = 1, \dots, k$, and $r_{i,j} \succ q_2$ for $i = 0, \dots, k$, $j = 1, \dots, i$. Thus

  $$t \to_s s\langle \vec{r}_0\rangle\langle t_1\langle \vec{r}_1\rangle, \dots, t_k\langle \vec{r}_k\rangle\rangle \succ q_0q_2(q_1q_2)$$

  and we may take $t' \equiv s\langle \vec{r}_0\rangle\langle t_1\langle \vec{r}_1\rangle, \dots, t_k\langle \vec{r}_k\rangle\rangle$.

If t is a tuple or q is not the contracted CL-pc-redex, then the claim follows from the inductive hypothesis. ◀

The following technical lemma shows that $\leftrightarrow_i$ may be postponed after $\to_s$.

▶ **Lemma 11.** *If $t \leftrightarrow_i \cdot \to_s^* t'$ then $t \to_s^* \cdot \leftrightarrow_i^{\equiv} t'$.*

Proof. Suppose $t_1 \leftrightarrow_i t_2 \to_s t_3$. We proceed by induction on the definition of $t_2 \to_s t_3$.

If t_2 is the contracted s-redex then, because an i-redex (i-contractum) is an i-term, it is easy to see by inspecting Definition 4 that the i-redex (i-contractum) in t_2 must occur below a variable position of the s-redex. Since significant reduction rules are linear and their conditions are stable under i-reductions (i-expansions), the claim holds. Note that we need $\leftrightarrow_i^{\equiv}$ instead of $\leftrightarrow_i$ in the conclusion, because the i-redex (i-contractum) may be erased by the s-contraction.

If t_2 is not the s-redex, then $t_2 \equiv s_1s_2$ or $t_2 \equiv \langle s_1, \dots, s_n\rangle$ with $n > 1$, and the claim is easily established possibly appealing to the inductive hypothesis. ◀

The next lemmas show that i-reductions/expansions and s-reductions preserve $\Downarrow_{\mathsf{F}_1}$.

▶ **Lemma 12.** *If t is standard and $t \leftrightarrow_i t'$ then t' is standard.*

Proof. We check that the conditions in Definition 8 hold for every subterm s' of t'. Note that because i-redexes and i-contracta are i-terms, s' is an i-term or there is a subterm s of t such that $s \leftrightarrow_i^{\equiv} s'$.

1. If s' is not an i-term, then there is a subterm s of t such that $s \leftrightarrow_i^{\equiv} s'$. If s is an i-term or a tuple then so is s'. Otherwise, s is an s-term by 1 in Definition 8. Then s' is also an s-term.
2. Suppose $s' \equiv \mathsf{C}_1t_0't_1't_2'$ with t_0' in s-NF. Since s' is not an i-term, there is a subterm s of t such that $s \equiv \mathsf{C}_1t_0t_1t_2$ and $t_i \leftrightarrow_i^{\equiv} t_i'$ for $i = 0, 1, 2$. Since t_0' is in s-NF and $t_0 \leftrightarrow_i^{\equiv} t_0'$, the term t_0 is also in s-NF. Thus $t_0 \equiv \mathsf{T}_1$ or $t_0 \equiv \mathsf{F}_1$ by 2 in Definition 8. Hence $t_0' \equiv \mathsf{T}_1$ or $t_0' \equiv \mathsf{F}_1$.
3. Suppose $s' \equiv \mathsf{C}_2t_0't_1't_2'$. Since s' is not an i-term, there is a subterm s of t such that $s \equiv \mathsf{C}_1t_0t_1t_2$ and $t_i \leftrightarrow_i^{\equiv} t_i'$ for $i = 0, 1, 2$. By 3 in Definition 8 we have $|t_1| =_{\text{CL-pc}^1} |t_2|$. Hence also $|t_1'| =_{\text{CL-pc}^1} |t_2'|$, because $t_i \leftrightarrow_i^{\equiv} t_i'$ implies $|t_i| =_{\text{CL-pc}^1} |t_i'|$.
4. Suppose $s' \equiv \mathsf{S}^{n_0,\dots,n_k}t_0't_1't_2'$. Since s' is not an i-term, there is a subterm s of t such that $s \equiv \mathsf{S}^{n_0,\dots,n_k}t_0t_1t_2$ and $t_i \leftrightarrow_i^{\equiv} t_i'$ for $i = 0, 1, 2$. By 4 in Definition 8 we conclude that t_2 is a tuple of length $n = \sum_{i=0}^{k} n_i$, and if $k > 1$ then t_1 is a tuple of length k. The same holds for t_2' and t_1', because a tuple cannot be an i-redex or an i-contractum.

5. Suppose s' is an s-term. There is a subterm s of t such that $s \leftrightarrow_i^{\equiv} s'$. Since s' is an s-term, so is s. Suppose $s' \to_s^* r'$. By Lemma 11 there is r such that $s \to_s^* r \leftrightarrow_i^{\equiv} r'$. By 5 in Definition 8, the term r is an s-term. Hence, r' is also an s-term.
6. Suppose $s' \equiv \langle t'_1, \ldots, t'_n \rangle$ with $n > 1$. Since s' is not an i-term, there is a subterm s of t such that $s \equiv \langle t_1, \ldots, t_n \rangle$ and $t_i \leftrightarrow_i^{\equiv} t'_i$ for $i = 1, \ldots, n$. By 6 in Definition 8 none of $t_1, \ldots, t_n$ is a tuple. Hence, none of $t'_1, \ldots, t'_n$ is a tuple either. ◀

▶ **Lemma 13.** *If $t \Downarrow_{\mathsf{F}_1}$ and $t \leftrightarrow_i t'$ then $t' \Downarrow_{\mathsf{F}_1}$.*

Proof. Suppose $t' \to_s^* t'_0$. By Lemma 11 there is t_0 with $t \to_s^* t_0$ and $t_0 \leftrightarrow_i^{\equiv} t'_0$. Because t is strongly standard, t_0 is standard. Hence t'_0 is standard by Lemma 12. Therefore t' is strongly standard.

Suppose $t' \to_s^* t'_0$ with t'_0 in s-NF. By Lemma 11 there is t_0 with $t \to_s^* t_0 \leftrightarrow_i^{\equiv} t'_0$. Since t'_0 is in s-NF, so is t_0, because an i-contraction or an i-expansion cannot create an s-redex. Since $t \Downarrow_{\mathsf{F}_1}$ we obtain $t_0 \equiv \mathsf{F}_1$. Thus $t'_0 \equiv t_0 \equiv \mathsf{F}_1$. ◀

▶ **Lemma 14.** *If $t \Downarrow_{\mathsf{F}_1}$ and $t \to_s t'$ then $t' \Downarrow_{\mathsf{F}_1}$.*

▶ **Corollary 15.** *If $t \Downarrow_{\mathsf{F}_1}$, $t \succ q$ and $q \to_{\text{CL-pc}} q'$ then there is t' with $t' \succ q'$ and $t' \Downarrow_{\mathsf{F}_1}$.*

Proof. Follows from Lemma 10, Lemma 13 and Lemma 14. ◀

With the above corollary we have finished the first half of the proof. Now we need to show an analogous corollary for CL-pc-expansions. First, we want to prove that CL-pc-expansions in unlabeled terms may be simulated by i-expansions and a-expansions in their strongly standard labeled variants. We have already shown in Lemma 13 that i-expansions preserve $\Downarrow_{\mathsf{F}_1}$. We need to show that a-expansions also preserve $\Downarrow_{\mathsf{F}_1}$.

One trivial reason why s-expansions do not necessarily preserve $\Downarrow_{\mathsf{F}_1}$ is that if $t \;{}_s\!\leftarrow t'$ then t' may be not standard even if t is, e.g., consider $\mathsf{F}_1 \;{}_s\!\leftarrow \mathsf{K}_1\mathsf{F}_1(\mathsf{CT}_1)$. A more profound reason is that with s-expansion we do not sufficiently "control" the expansion by a rule for C_2. E.g. $\mathsf{F}_1 \;{}_s\!\leftarrow \mathsf{C}_2\Omega\mathsf{F}_1(\mathsf{KF}\Omega)$. Then $\mathsf{C}_2\Omega\mathsf{F}_1(\mathsf{KF}\Omega) \to_s \mathsf{KF}\Omega$ but $\mathsf{KF}\Omega$ does not s-reduce to F_1.

Hence, we use a-expansions which put additional restrictions on the s-redexes, essentially implementing the labeling of expansions described in Section 2. They also allow to "delay" the reductions in a contractum of $\mathsf{C}_2 t_0 t_1 t_1$ to facilitate the proof of an analogon of Lemma 11.

Like in the proof of Lemma 13 we show that if $t' \to_a t$ then any reduction $t' \to_s^* s'$ may be simulated by a reduction $t \to_s^* s$ with $s' \to_a^{\equiv} s$. The most interesting case is when $t' \equiv E[\mathsf{C}_2 t_0 t_1 t_1] \to_a E[t_1] \equiv t$ (where E is a context), which is obtained from a CL-pc-expansion by the rule $\mathsf{C}xyy \to y$. We now informally describe the idea for the proof in this case. Thus suppose $t' \to_s^* s'$. If a contracted s-redex does not overlap with a descendant[5] of $\mathsf{C}_2 t_0 t_1 t_1$, then the s-reduction is simulated by the same s-reduction. If a descendant of $\mathsf{C}_2 t_0 t_1 t_1$ occurs inside a contracted s-redex, but it is different from this redex, then the descendant must occur below a variable position of the s-redex, because there are no non-root overlaps between the rules of significant reduction. Thus we may simulate this s-reduction by the same s-reduction. If a contracted s-redex occurs inside a descendant $\mathsf{C}_2 t'_0 t'_1 t'_2$ of $\mathsf{C}_2 t_0 t_1 t_1$, but it is different from this descendant, then it must occur in t'_0, t'_1 or t'_2. In this case we ignore the s-contraction while at all times maintaining the invariant: if $\mathsf{C}_2 t'_0 t'_1 t'_2$ is a descendant of $\mathsf{C}_2 t_0 t_1 t_1$ then $t_1 \to_s^* t'_1$ and $t_1 \to_s^* t'_2$, and the descendant of t_1 in the simulated reduction is always identical with t_1, i.e. t_1 (the a-contractum of $\mathsf{C}_2 t_0 t_1 t_1$) is not changed by

[5] Note that because the rules of significant reduction are linear there may be at most one descendant.

the simulated s-reduction. Finally, if a descendant $\mathsf{C}_2t_0't_1't_2'$ of $\mathsf{C}_2t_0t_1t_1$ is s-contracted, then either $\mathsf{C}_2t_0't_1't_2' \to_s t_1'$ or $\mathsf{C}_2t_0't_1't_2' \to_s t_2'$. In any case we can s-reduce t_1 to t_1' or t_2'. In other words, we defer the choice of the simulated reduction path till the descendant of the a-redex is actually contracted.

▶ **Definition 16.** An l-term t' is an *a-redex* and t its *a-contractum*, if t is an s-term and one of the following holds:

- $t' \equiv \mathsf{C}_1\mathsf{T}_1tq$ and q is an i-term,
- $t' \equiv \mathsf{C}_1\mathsf{F}_1qt$ and q is an i-term,
- $t' \equiv \mathsf{C}_2qt_1t_2$, $t \to_s^* t_1$, $t \to_s^* t_2$ and q is an i-term,
- $t' \equiv \mathsf{K}_1tq$ and q is an i-term,
- $t' \equiv \mathsf{S}^{\vec{n}}t_0\langle s_1,\dots,s_k\rangle\langle\vec{r}_0,\dots,\vec{r}_k\rangle$ where the conventions regarding vector notation are as in Definition 4, $|s_i| =_{\text{CL-pc}^1} |s_j|$ for $i,j = 1,\dots,k$, $|r_{i,j}| =_{\text{CL-pc}^1} |r_{i',j'}|$ for $i,i' = 0,\dots,k$, $j = 1,\dots,n_i$, $j' = 1,\dots,n_{i'}$, none of the s_i or $r_{i,j}$ is a tuple, and $t \equiv t_0\langle\vec{r}_0\rangle\langle s_1\langle\vec{r}_1\rangle,\dots,s_k\langle\vec{r}_k\rangle\rangle$.

Because of the third point, an a-contractum of an a-redex is not unique. The notations $\to_a$, $\to_a^*$, $\to_{i,a}$, etc. are used accordingly. Note that any a-redex is an s-redex.

▶ **Lemma 17.** *If $t' \to_a t$ then $t' \to_s \cdot\, {}_s^*\!\leftarrow t$, and hence $|t'| =_{\text{CL-pc}^1} |t|$.*

The above simple lemma implies that the conditions in significant reduction rules are stable under a-reduction and a-expansion. Note that if $t' \to_a t$ then not necessarily $t' \to_s t$ because of the third point in Definition 16.

▶ **Lemma 18.** *If t is standard, $t \succ q$ and q $_{\text{CL-pc}}\!\leftarrow q'$ then there is t' with $t' \to_{i,a}^* t$ and $t' \succ q'$.*

Proof. Induction on the size of t. First assume t is not a tuple and q is the CL-pc-contractum expanded in q $_{\text{CL-pc}}\!\leftarrow q'$. If t is an i-term, then $t \equiv q$ $_i\!\leftarrow q'$ and we may take $t' \equiv q'$. If t is not an i-term, then it is an s-term by 1 in Definition 8. We have the following possibilities, depending on the rule of CL-pc used in the expansion.

- If $q' \equiv \mathsf{CT}q_1q_2 \to_{\text{CL-pc}} q_1 \equiv q$ then we take $t' \equiv \mathsf{C}_1\mathsf{T}_1tq_2$ and we have $t' \to_a t$ and $t' \succ q'$.
- If $q' \equiv \mathsf{CF}q_1q_2 \to_{\text{CL-pc}} q_2 \equiv q$ then we may take $t' \equiv \mathsf{C}_1\mathsf{F}_1q_1t$.
- If $q' \equiv \mathsf{C}q_0q_1q_1 \to_{\text{CL-pc}} q_1$ then we may take $t' \equiv \mathsf{C}_2q_0tt$.
- If $q' \equiv \mathsf{K}q_0q_1 \to_{\text{CL-pc}} q_0$ then we may take $t' \equiv \mathsf{K}_1tq_1$.
- If $q' \equiv \mathsf{S}q_0q_1q_2 \to_{\text{CL-pc}} q_0q_2(q_1q_2)$ then $t \succ q_0q_2(q_1q_2)$ and t is an s-term. Hence $t \equiv t_at_bt_c$ with $t_a \succ q_0$, $t_b \succ q_2$ and $t_c \succ q_1q_2$. Recalling the convention $\langle s\rangle \equiv s$ for any term s, we may assume

$$t_b \equiv \langle s_1,\dots,s_m\rangle, t_c \equiv \langle t_1,\dots,t_k\rangle, \text{ for } k,m \in \mathbb{N}_+,$$
$$\text{if } k = 1 \text{ then } t_1 \text{ is not a tuple, and if } m = 1 \text{ then } s_1 \text{ is not a tuple.} \qquad (\star)$$

In other words, if e.g. t_b is a tuple, then $t_b \equiv \langle s_1,\dots,s_m\rangle$ for some $s_1,\dots,s_m$. If t_b is not a tuple then we take $s_1 \equiv t_b$ and consider $t_b \equiv \langle t_b\rangle \equiv \langle s_1\rangle$. This is chiefly to reduce the number of cases to consider. Let $1 \le i \le k$. Because $t_b \succ q_2$, we have $s_i \succ q_2$ for $i = 1,\dots,m$. Also none of $s_1,\dots,s_m$ is a tuple, by condition 6 in Definition 8, or by $(\star)$ if $m = 1$. Since $t_c \succ q_1q_2$, we have $t_i \succ q_1q_2$. Also t_i cannot be a tuple, by condition 6 in Definition 8, or by $(\star)$ if $k = 1$. Thus $t_i \equiv u_i\langle\vec{r}_i\rangle$ where $\vec{r}_i$ stands for $r_{i,1},\dots,r_{i,n_i}$, and $u_i \succ q_1$ and $r_{i,j} \succ q_2$ for $j = 1,\dots,n_i$, where none of the $r_{i,j}$ is a tuple, by definition (if $n_i = 1$) or by condition 6 in Definition 8. By Lemma 9 also none of $u_1,\dots,u_k$ is a tuple. We may thus take $t' \equiv \mathsf{S}^{m,n_1,\dots,n_k}t_a\langle u_1,\dots,u_k\rangle\langle\vec{r}_0,\vec{r}_1,\dots,\vec{r}_k\rangle$ where $\vec{r}_0$ stands for $s_1,\dots,s_m$. We have $t' \to_a t$ and $t' \succ q'$.

If t is a tuple or q is not the CL-pc-contractum, then the claim follows from the inductive hypothesis. ◀

▶ **Lemma 19.** *If $t\ {}_a\!\leftarrow \cdot \to_s t'$ and t is standard then $t \to_s^* \cdot \overset{\equiv}{}_a\!\leftarrow t'$.*

Proof. Suppose $t' \to_s t'_1$, $t' \to_a t$ and t is standard. By induction on the definition of $t' \to_s t'_1$ we show that there is t_1 with $t \to_s^* t_1$ and $t'_1 \to_a^{\equiv} t_1$. The base case is when the s-contraction in $t' \to_s t'_1$ occurs at the root.

If the s-contraction occurs at the root, but the a-contraction in $t' \to_a t$ does not occur at the root, then it is easy to see by inspecting the definitions that the a-redex in t'_1 must occur below a variable position of the s-redex. Since significant reduction rules are linear and their conditions are stable under a-reduction, the claim holds in this case.

Assume that both the s-contraction and the a-contraction occur at the root. If $t' \equiv \mathsf{C}_2 q s_1 s_2 \to_a t$ then $t \to_s^* s_1$, $t \to_s^* s_2$ and the s-contraction of t' yields either s_1 or s_2. We may thus take either $t_1 \equiv s_1$ or $t_1 \equiv s_2$, and we have $t \to_s^* t_1 \equiv t'_1$. If $t' \equiv \mathsf{C}_1\mathsf{T}_1 t q \to_a t$ then the s-contraction must be by the first rule of CL-pc^s, so $t'_1 \equiv t$ and we may take $t_1 \equiv t'_1 \equiv t$. All other cases are analogous.

If neither the s-contraction nor the a-contraction occurs at the root, then the claim is easily established, possibly appealing to the inductive hypothesis.

Finally, assume that the a-contraction occurs at the root, but the s-contraction does not occur at the root. We have the following possibilities.

- If $t' \equiv \mathsf{C}_1\mathsf{T}_1 t q \to_a t$ then the s-contraction must occur inside t. So $t \to_s t_1$ for some term t_1. Note that t is an s-term by definition of a-contraction. Therefore t_1 is also an s-term, by 5 in Definition 8. Thus t_1 satisfies the required conditions.
- If $t' \equiv \mathsf{C}_2 q s_1 s_2 \to_a t$ then $t \to_s^* s_1$, $t \to_s^* s_2$ and the s-contraction must occur inside s_1 or s_2. We may take $t_1 \equiv t$ and we still have $t'_1 \to_a t_1$.
- The cases $t' \equiv \mathsf{C}_1\mathsf{F}_1 q t \to_a t$ and $t' \equiv K_1 t q \to_a t$ are analogous to the first case.
- If $t' \equiv \mathsf{S}^{\vec{n}} t_0\langle s_1, \dots, s_k\rangle\langle \vec{r}_0, \dots, \vec{r}_k\rangle$ then $|s_i| =_{\text{CL-pc}^1} |s_j|$, $|r_{i,j}| =_{\text{CL-pc}^1} |r_{i',j'}|$ for i, j, i', j' as in Definition 16, none of the s_i or $r_{i,j}$ is a tuple, and $t \equiv t_0\langle \vec{r}_0\rangle\langle s_1\langle \vec{r}_1\rangle, \dots, s_k\langle \vec{r}_k\rangle\rangle$. The s-contraction must occur inside one of the s_i or the $r_{i,j}$, or in t_0. For instance, assume $s_1 \to_s s'_1$. Since s_1 is a subterm of t and it is not a tuple, it cannot s-reduce to a tuple by Definition 8. Hence s'_1 is not a tuple. Take $t_1 \equiv t_0\langle \vec{r}_0\rangle\langle s'_1\langle \vec{r}_1\rangle, s_2\langle \vec{r}_2\rangle, \dots, s_k\langle \vec{r}_k\rangle\rangle$. Note that $t \to_s t_1$. Thus t_1 is an s-term, because t is an s-term and it s-reduces only to s-terms, by 5 in Definition 8. ◀

▶ **Corollary 20.** *If $t\ {}_a\!\leftarrow \cdot \to_s^* t'$ and t is strongly standard then $t \to_s^* \cdot \overset{\equiv}{}_a\!\leftarrow t'$.*

▶ **Lemma 21.** *If r is a strongly standard a-contractum of an a-redex r', and s' is a proper subterm of r', then s' is standard.*

Proof. It suffices to show that s' is a subterm of some standard term.

- Suppose $r' \equiv \mathsf{C}_1\mathsf{T}_1 r q \to_a r$ with q an i-term. Both $\mathsf{C}_1\mathsf{T}_1 r$ and q are standard and s' is a subterm of one of them. The cases $r' \equiv \mathsf{C}_1\mathsf{F}_1 q r$ and $r' \equiv \mathsf{K}_1 r q$ are analogous.
- Suppose $r' \equiv \mathsf{C}_2 q r_1 r_2 \to_a r$ with q an i-term. Because $r \to_s^* r_i$ and r is strongly standard, r_1, r_2 are standard. Also q is an i-term. This implies that $\mathsf{C}_2 q r_1$ is also standard. Since s' occurs in $\mathsf{C}_2 q r_1$ or r_2, it is standard.
- Suppose $r' \equiv \mathsf{S}^{\vec{n}} t_0\langle s_1, \dots, s_k\rangle\langle \vec{r}_0, \dots, \vec{r}_k\rangle \to_a t_0\langle \vec{r}_0\rangle\langle s_1\langle \vec{r}_1\rangle, \dots, s_k\langle \vec{r}_k\rangle\rangle$. The term t_0 and each of s_i and $r_{i,j}$ (with i, j as in Definition 16) is standard. Note that none of s_i or $r_{i,j}$ is a tuple by Definition 16. Since each s_i is also standard, by inspecting Definition 8 we may conclude that $\langle s_1, \dots, s_k\rangle$ is standard. Similarly $\langle \vec{r}_0, \dots, \vec{r}_k\rangle$ is standard. Also

$\mathsf{S}^{\vec{n}}t_0\langle s_1,\dots,s_k\rangle$ is standard. This implies that s' is standard, because it occurs in $\mathsf{S}^{\vec{n}}t_0\langle s_1,\dots,s_k\rangle$ or $\langle \vec{r}_0,\dots,\vec{r}_k\rangle$. ◀

▶ **Lemma 22.** *If s is an s-term and $s' \to_a s$ then s' is also an s-term.*

Proof. Induction on the structure of s. ◀

▶ **Lemma 23.** *If t is strongly standard and $t' \to_a t$ then t' is standard.*

Proof. We check that the conditions in Definition 8 hold for every subterm s' of t'. We may assume that s' does not occur in t, as otherwise the claim follows from the fact that t is standard. Therefore, s' occurs in the a-redex contracted in $t' \to_a t$, or the a-redex occurs inside s'. If s' is a proper subterm of the a-redex, then our claim holds by Lemma 21. Hence, we may assume that the a-redex r' is a subterm of s'. Then $s' \to_a s$ with s a subterm of t (so s is strongly standard).

1. Suppose r is the a-contractum of r' and $s' \to_a s$. By Definition 16, the term r is an s-term. Thus s cannot be an i-term. If s is a tuple, then so is s'. Otherwise, s is an s-term, by 1 in Definition 8. Hence s' is also an s-term by Lemma 22.
2. Suppose $s' \equiv \mathsf{C}_1t'_0t'_1t'_2$ and t'_0 is in s-NF. If $s' \equiv r'$ then $s' \equiv \mathsf{C}_1\mathsf{T}_1t'_1t'_2$ or $s' \equiv \mathsf{C}_1\mathsf{F}_1t'_1t'_2$, hence $t'_0 \equiv \mathsf{T}_1$ or $t'_0 \equiv \mathsf{F}_1$. If r' is a proper subterm of s', then r' must be a subterm of t'_1 or t'_2, because a-redexes are not in s-NF. Thus, $s' \to_a s \equiv \mathsf{C}_1t'_0t_1t_2$ for some terms t_1, t_2, where s is a subterm of t. Hence, $t'_0 \equiv \mathsf{T}_1$ or $t'_0 \equiv \mathsf{F}_1$ by 2 in Definition 8.
3. Suppose $s' \equiv \mathsf{C}_2t'_0t'_1t'_2$. If $s' \equiv r'$ then $s \to_s^* t'_1$ and $s \to_s^* t'_2$. Hence $|t'_1| =_{\text{CL-pc}^1} |s| =_{\text{CL-pc}^1} |t'_2|$. If $s' \not\equiv r'$ then $s \equiv \mathsf{C}_2t_0t_1t_2$ with $t'_i \to_a^{\equiv} t_i$. Because s is standard, $|t_1| =_{\text{CL-pc}^1} |t_2|$ by 3 in Definition 8. Thus also $|t'_1| =_{\text{CL-pc}^1} |t'_2|$ by Lemma 17.
4. Suppose $s' \equiv \mathsf{S}^{n_0,\dots,n_k}t'_0t'_1t'_2$. If $s' \equiv r'$, then $s' \equiv \mathsf{S}^{n_0,\dots,n_k}t'_0\langle s_1,\dots,s_k\rangle\langle \vec{r}_0,\dots,\vec{r}_k\rangle$, as in Definition 16, so the claim holds. If r' is a proper subterm of s', then $s' \equiv \mathsf{S}^{n_0,\dots,n_k}t'_0t'_1t'_2 \to_a s \equiv \mathsf{S}^{n_0,\dots,n_k}t_0t_1t_2$ where $t'_i \to_a^{\equiv} t_i$ for $i = 0, 1, 2$, and s is a subterm of t. By 4 in Definition 8, the term t_2 is a tuple of length $\sum_{i=0}^{k} n_i$, and if $k > 1$ then t_1 is a tuple of length k. Since an a-contractum is an s-term, and hence not a tuple, t_2 is not an a-contractum, and if $k > 1$ then t_1 is not an a-contractum. Thus we may conclude that t'_2 is a tuple of length $\sum_{i=0}^{k} n_i$, and if $k > 1$ then t'_1 is a tuple of length k.
5. Suppose s' is an s-term and $s' \to_s^* s'_1$. Because also $s' \to_a s$ and s is strongly standard, by Corollary 20 there is s_1 with $s'_1 \to_a^{\equiv} s_1$ and $s \to_s^* s_1$. By Definition 16, the term s is an s-term, so s_1 is also an s-term by 5 in Definition 8. So s'_1 is an s-term by Lemma 22.
6. Suppose $s' \equiv \langle t'_1,\dots,t'_n\rangle$ with $n > 1$. We have $s' \to_a s \equiv \langle t_1,\dots,t_n\rangle$ where $t'_i \to_a^{\equiv} t_i$ for $i = 1,\dots,n$. By 6 in Definition 8, none of $t_1,\dots,t_n$ is a tuple. Thus it is easy to see by inspecting Definition 16 that none of $t'_1,\dots,t'_n$ can be a tuple. ◀

▶ **Lemma 24.** *If $t \Downarrow_{\mathsf{F}_1}$ and $t' \to_a t$ then $t' \Downarrow_{\mathsf{F}_1}$.*

Proof. Suppose $t' \to_s^* t'_0$. By Corollary 20 there is t_0 with $t \to_s^* t_0$ and $t'_0 \to_a^{\equiv} t_0$. Since t is strongly standard, so is t_0. Therefore, t'_0 is standard by Lemma 23.

Suppose $t' \to_s^* t'_0$ with t'_0 in s-NF. By Corollary 20 there is t_0 with $t \to_s^* t_0$ and $t'_0 \to_a^{\equiv} t_0$. Since an a-redex is an s-redex and t'_0 is in s-NF, we conclude that $t'_0 \equiv t_0$. But then $t'_0 \equiv t_0 \equiv \mathsf{F}_1$, because $t \Downarrow_{\mathsf{F}_1}$. ◀

▶ **Corollary 25.** *If $t \Downarrow_{\mathsf{F}_1}$, $t \succ q$ and $q \ {}_{\text{CL-pc}}\!\leftarrow q'$ then there is t' with $t' \Downarrow_{\mathsf{F}_1}$ and $t' \succ q'$.*

Proof. Follows from Lemma 18, Lemma 13 and Lemma 24. ◀

▶ **Lemma 26.** *If t has no s-NFs other than F_1 then $t \to_{s-}^* \mathsf{F}_1$.*

Proof. Since s-reduction is terminating, by reducing s-redexes outside any tuples and not using the second rule for C_2 we will ultimately obtain a term t' with all s-redexes inside tuples, and such that $t \to^*_{s-} t'$. Note that an s-redex in t' may only occur inside a tuple, because any s-redex by the second rule for C_2 is also an $s-$-redex by the first rule for C_2. If t' is in s-NF then $t' \equiv \mathsf{F}_1$. Otherwise, any s-NF of t' must contain a tuple, because s-reduction inside a tuple cannot erase this tuple or create an s-redex outside of it. But since any s-NF of t' is an s-NF of t, this contradicts the fact that t has no s-NFs other than F_1. ◀

We now have everything we need to show the central lemma of the confluence proof.

▶ **Lemma 27.** *The system* CL-pc^1 *is* F*-normal, i.e., if* $q =_{\text{CL-pc}^1} \mathsf{F}$ *then* $q \to^*_{\text{CL-pc}^1} \mathsf{F}$.

Proof. If $q =_{\text{CL-pc}^1} \mathsf{F}$ then by Lemma 2 we have $q =_{\text{CL-pc}} \mathsf{F}$. Note that $\mathsf{F}_1 \Downarrow_{\mathsf{F}_1}$ and $|\mathsf{F}_1| \equiv \mathsf{F}$. Thus, using Corollary 15 and Corollary 25 it is easy to show by induction on the length of $q =_{\text{CL-pc}} \mathsf{F}$ that there is t with $t \succ q$ and $t \Downarrow_{\mathsf{F}_1}$. By Lemma 26 we have $t \to^*_{s-} \mathsf{F}_1$. But then, because $t \succ q$, using Lemma 7 it is easy to show by induction on the length of $t \to^*_{s-} \mathsf{F}_1$ that $q \equiv |t| \to^*_{\text{CL-pc}^1} |\mathsf{F}_1| \equiv \mathsf{F}$. ◀

It remains to derive the confluence of CL-pc^1 and CL-pc$^{\text{L}}$ from Lemma 27. We use a trick with an auxiliary term rewriting system R, in a way similar to how the confluence of CL-pc$^{\text{L}}$ is derived from the condition $\mathsf{T} \neq_{\text{CL-pc}^{\text{L}}} \mathsf{F}$ in [4]. The idea is to eliminate the non-trivial overlap between the rules of CL-pc^1 by imposing additional side conditions.

▶ **Definition 28.** The term rewriting system R is defined by the following rules:

$$\begin{array}{lcl@{\qquad}lcll} \mathsf{K}xy & \to & x & \mathsf{CT}xy & \to & x & \\ \mathsf{S}xyz & \to & xz(yz) & \mathsf{C}zxy & \to & y & \Leftarrow \quad z =_{\text{CL-pc}^1} \mathsf{F} \\ & & & \mathsf{C}zxy & \to & x & \Leftarrow \quad z \neq_{\text{CL-pc}^1} \mathsf{F} \wedge x =_{\text{CL-pc}^1} y \end{array}$$

▶ **Lemma 29.** *If* $q \to_{\text{CL-pc}} q'$ *then* $q \to_R q'$.

▶ **Lemma 30.** *If* $q \to_R q'$ *then* $q \to^*_{\text{CL-pc}^1} q'$.

Proof. Follows from definitions and Lemma 27. ◀

▶ **Lemma 31.** *The system* R *is confluent.*

Proof. Because $\mathsf{T} \neq_{\text{CL-pc}^1} \mathsf{F}$ by Lemma 27, the system R is weakly orthogonal (i.e. it is left-linear and all its critical pairs are trivial). By Lemma 30 the conditions are stable under reduction. Weakly orthogonal conditional term rewriting systems whose conditions are stable under reduction are confluent [11, Chapter 4]. ◀

▶ **Theorem 32.** *The systems* CL-pc^1 *and* CL-pc$^{\text{L}}$ *are confluent.*

Proof. Since $q_1 \to_{\text{CL-pc}^1} q_2$ implies $q_1 \to_{\text{CL-pc}^{\text{L}}} q_2$, it suffices to show that if $q_1 =_{\text{CL-pc}^{\text{L}}} q_2$ then there is q with $q_1 \to^*_{\text{CL-pc}^1} q$ and $q_2 \to^*_{\text{CL-pc}^1} q$. So suppose $q_1 =_{\text{CL-pc}^{\text{L}}} q_2$. Then by Lemma 2 we have $q_1 =_{\text{CL-pc}} q_2$. By Lemma 29 we obtain $q_1 =_R q_2$. By Lemma 31 there is q with $q_1 \to^*_R q$ and $q_2 \to^*_R q$. By Lemma 30 we have $q_1 \to^*_{\text{CL-pc}^1} q$ and $q_2 \to^*_{\text{CL-pc}^1} q$. ◀

References

1 F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1999.

2 P. Chew. Unique normal forms in term rewriting systems with repeated variables. In *STOC '81*, pages 7–18. ACM, 1981.

3 Ł. Czajka and C. Kaliszyk. Hammer for Coq: Automation for dependent type theory. Submitted. Available at `http://cl-informatik.uibk.ac.at/cek/coqhammer/`, 2017.

4 R. C. de Vrijer. Conditional linearization. *Indagationes Mathematicae*, 10(1):145–159, 1999.

5 N. Dershowitz, J.-P. Jouannaud, and J. W. Klop. Open problems in rewriting. In *RTA '91*, pages 445–456, 1991.

6 S. Kahrs and C. Smith. Non-omega-overlapping TRSs are UN. In *FSCD 2016*, pages 22:1–22:17, 2016.

7 J. W. Klop. *Combinatory reduction systems*, volume 127 of *Mathematical Centre Tracts*. Amsterdam, 1980.

8 J. W. Klop and R. C. de Vrijer. Unique normal forms for lambda calculus with surjective pairing. *Inf. and Comp.*, 80(2):97–113, 1989.

9 K. Mano and M. Ogawa. Unique normal form property of compatible term rewriting systems: a new proof of Chew's theorem. *Theor. Comp. Sci.*, 258(1):169–208, 2001.

10 K. Støvring. Extending the extensional lambda calculus with surjective pairing is conservative. *Logical Methods in Computer Science*, 2(2), 2006.

11 Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.

12 Y. Toyama and M. Oyamaguchi. Church-Rosser property and unique normal form property of non-duplicating term rewriting systems. In *CTRS-94*, pages 316–331, 1995.

13 R. Verma. Unique normal forms for nonlinear term rewriting systems: Root overlaps. In *FCT '97*, pages 452–462. Springer, 1997.

The Complexity of Principal Inhabitation

Andrej Dudenhefner[1] **and Jakob Rehof**[2]

1 **Department of Computer Science, Technical University of Dortmund, Dortmund, Germany**
andrej.dudenhefner@cs.tu-dortmund.de

2 **Department of Computer Science, Technical University of Dortmund, Dortmund, Germany**
jakob.rehof@cs.tu-dortmund.de

Abstract

It is shown that in the simply typed λ-calculus the following decision problem of *principal inhabitation* is PSPACE-complete: Given a simple type τ, is there a λ-term N in β-normal form such that τ is the principal type of N?

While a Ben-Yelles style algorithm was presented by Broda and Damas in 1999 to count normal principal inhabitants (thereby answering a question posed by Hindley), it does not induce a polynomial space upper bound for principal inhabitation. Further, the standard construction of the polynomial space lower bound for simple type inhabitation does not carry over immediately.

We present a polynomial space bounded decision procedure based on a characterization of principal inhabitation using path derivation systems over subformulae of the input type, which does not require candidate inhabitants to be constructed explicitly. The lower bound is shown by reducing a restriction of simple type inhabitation to principal inhabitation.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Lambda Calculus, Type Theory, Simple Types, Inhabitation, Principal Type, Complexity

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.15

1 Introduction and Related Work

The inhabitation problem for simply typed λ-calculus [1] (given a type, is there a λ-term having the type?) is known to be PSPACE-complete by a well-known result of Statman [11]. Due to the subject reduction and normalization theorems for simple types, it is sufficient to decide the existence of inhabitants in β-normal form. A natural related problem is the problem of *principal inhabitation*: Given a type τ, is there a *normal principal inhabitant* of τ? A normal principal inhabitant of τ is a λ-term in β-normal form having τ as its principal type [8, Definition 8A11]. The principal inhabitation problem is different from the inhabitation problem. For, whereas every inhabited type τ is also the principal type of *some* λ-term [8, Lemma 7A2 (i)], this is not the case when we restrict attention to inhabitants in β-normal form: Some inhabited types are not principally inhabited, because they are not the principal types of any β-normal form. For example, $\tau = a \to a \to a$ is inhabited by $K \equiv \lambda x.\lambda y.x$, but τ is not the principal type of K (its principal type is $a \to b \to a$). In fact, there is no β-normal form having τ as its principal type (cf. [8, Remark 8A13 (iii)]), therefore τ is not principally inhabited. Since normal principal inhabitants can be seen as natural implementations of a given type specification in the context of type-based program synthesis [9, 6, 3], principal inhabitation is not only of systematic but also of practical importance.

In this paper we are concerned with the complexity of the principal inhabitation problem. The only known results directly related to the upper bound for principal inhabitation are the counting procedure by Broda and Damas [4] and its generalization by principal proof trees in [5]. Broda and Damas present in [4] a Ben-Yelles style counting algorithm [2, 8] for normal principal inhabitants, thereby solving a problem mentioned by Hindley in [8, Problem 8D10 (i)]. In [5], a more general technique (formula-tree proof method) is used to construct so-called principal proof trees deciding principal inhabitation. However, these results do not immediately imply a polynomial space upper bound for principal inhabitation. In particular, the counting procedure of [4] operates by explicitly enumerating inhabitants and checking for principality in each case. Although a depth-bound is provided on inhabitant terms, which is polynomial in the size of the input type τ, inhabitants may be of *exponential size*. Therefore, the upper bound for principal inhabitation induced by the procedure is exponential time. Because principality is a *global property* of a derivation and is therefore sensitive to the exact structure of inhabitants, it does not appear to be obvious how to obviate an exponential time construction. Similarly to [4], principal proof trees in [5] can be of exponential size, inducing a similar complexity as the previous approach. Further remarks comparing details of our decision procedure with the approaches in [4, 5] can be found within the technical development of the paper.

In comparison with the inhabitation problem for simple types, basic challenges for a polynomial space upper bound for principal inhabitation include the following two complications. For one, it is not possible to bound the size of the type environment during inhabitant search by simply coalescing type variables having the same type. In the standard approach [12], when searching for a long normal inhabitant[1] of a function type $\sigma \to \tau$, an assumption $(x : \sigma)$ is added to the environment only if it does not already contain some variable of type σ. This leads to a linear upper bound on environment size, because (as a consequence of the subformula property for normal forms) only subformulae of the original input need to be considered during inhabitant search. However, this approach leads to an incomplete procedure for principal inhabitation. Consider as an example principally inhabiting $(a \to a \to a) \to a \to a \to a$. The procedure would (implicitly) discover a λK-term $\lambda f.\lambda x.\lambda y.f(fxx)(fxx)$ as inhabitant in which the fifth (second from right) occurrence of a is implicitly associated with a variable y which is not used, because it is coalesced with x (also of type a) in the body of the term. But this term is not a principal inhabitant, whereas $\lambda f.\lambda x.\lambda y.f(fxy)(fyx)$ is. In essence, the solution to this problem for principal inhabitation lies in only coalescing type assumptions associated with the same subformula *occurrence* in the goal type. This approach is realized in our solution by keeping track of such occurrences and relations between them using a calculus of paths (subformula calculus) which distinguishes subformula occurrences.

The second complication in comparison with the standard procedure has to do with certain kinds of cyclic situations. The alternating search procedure of [12] does not need to inhabit a goal which has already appeared under the same assumptions on the current branch of the search tree, but such a strategy would be incomplete for principal inhabitation. To illustrate, consider the type $\tau \equiv (a \to a) \to a \to a$. It is inhabited by every Church numeral, but not principally so. Whereas the standard procedure would determine the inhabitant $\mathbf{c}_0 = \lambda f.\lambda x.x$, only the Church numerals $\mathbf{c}_n$ for $n \geq 2$ are normal principal inhabitants of τ. For example, $\mathbf{c}_2 = \lambda f.\lambda x.f(fx)$ is a normal principal inhabitant of τ, because the cyclic proof structure – proving inhabitation of a by (fx), although there is already an

[1] By a long normal inhabitant is meant a λ-term in η-long β-normal form, see [8, Definition 8A7].

inhabitant, x, of a in a subexpression (premise) – forces identification of the domain and range types of f. This phenomenon can apparently get more complicated. For example, the type $(a \to a) \to (a \to a) \to a \to a$ is principally inhabited by the term $\lambda f.\lambda g.\lambda x.f(g(g(fx)))$, but neither by $\lambda f.\lambda g.\lambda x.g(g(g(gx)))$ nor by $\lambda f.\lambda g.\lambda x.f(g(f(gx)))$. The complications arising from this phenomenon are handled by path deduction systems characterizing exactly the necessary and sufficient identifications among subformula occurrences of types without explicit reference to inhabitant terms. An important instrument to this end is an adaptation of the *subformula filtration* technique, which was introduced in [7] for the intersection type system.

To provide an upper bound, we present a polynomial space bounded decision procedure based on a characterization of principal inhabitation using a calculus over subformulae of the input type, which does not require candidate inhabitants to be constructed explicitly.

With regard to the lower bound, one cannot directly transfer the polynomial space lower bound for the inhabitation problem [11, 12], because it turns out (as will be shown) that the standard reduction (cf. [12]) from truth of quantified Boolean formulae uses types which are not necessarily principally inhabited. However, we observe that the standard reduction induces a PSPACE-hard restriction of simple type inhabitation. Therefore, for the polynomial space lower bound, we reduce this particular restriction to principal inhabitation.

The paper is organized as follows. After preliminary definitions (Section 2) we introduce (Section 3) subformula filtration to obtain a necessary condition (Lemma 14) on the form of type derivations for principal inhabitants, which will be of pervasive importance in the paper. We then (Section 4) define the subformula calculus, which allows us to talk about subformula occurrences and relations between them in type derivations to characterize principal inhabitants (Theorem 32). In Section 5 we present the algorithm (INH) to decide principal inhabitation and prove the polynomial space upper bound. The proof of the PSPACE lower bound is given in Section 6. We conclude the paper in Section 7 which also contains remarks about future work.

2 Simply-Typed Lambda Calculus

In this section we briefly assemble the necessary prerequisites in order to discuss principal inhabitation in the simply typed λ-calculus. We denote λ-terms (cf. Definition 1) by L, M, N and simple types (cf. Definition 2) are denoted by σ, τ, ρ, where type atoms are denoted by a, b, c and drawn from the denumerable set $\mathbb{A}$. The rules (Ax), ($\to$I) and ($\to$E) of the simple type system are given in Definition 3.

▶ **Definition 1** (λ-Terms). $L, M, N ::= x \mid (\lambda x.M) \mid (M\ N)$.

▶ **Definition 2** (Simple Types). $\sigma, \tau, \rho ::= a \mid \sigma \to \tau$ where $a \in \mathbb{A}$.

▶ **Definition 3** (Simple Type System).

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\ (\text{Ax}) \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\ (\to\text{I}) \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\ N : \tau}\ (\to\text{E})$$

We write $\mathcal{D} \triangleright \Gamma \vdash M : \tau$, if the derivation $\mathcal{D}$ derives the judgement $\Gamma \vdash M : \tau$, i.e. $\mathcal{D}$ is a finite tree of judgements with root $\Gamma \vdash M : \tau$ that respects the corresponding typing rules.

Type substitutions (cf. [8, Definition 3A1]) are denoted by S and are lifted from type atoms to types. A principal type (cf. Definition 4) of a term is the most general type that can be assigned to that term and is unique up to atom renaming. A normal principal inhabitant (cf. Definition 5) is a closed β-normal form for which the given type is principal.

▶ **Definition 4** (Principal Type)**.** We say that τ is a *principal type* of M, if $\vdash M : \tau$ and for all types σ such that $\vdash M : \sigma$ there exists a substitution S such that $S(\tau) = \sigma$.

▶ **Definition 5** (Normal Principal Inhabitant)**.** We say that a λ-term M in β-normal form is a *normal principal inhabitant* of τ, if τ is the principal type of M.

As usual, term application is left-associative. In accordance with [8], we define $\mathrm{Long}(\tau)$ as the set of all long normal inhabitants of τ.

▶ **Definition 6** ($\mathrm{Long}(\tau)$)**.** The set $\mathrm{Long}(\tau)$ consists of all λ-terms M such that $\vdash M : \tau$ is derivable using only the rule ($\to$I) and the following rule ($\to_L$E)

$$\frac{\Gamma, x : \sigma_1 \to \ldots \to \sigma_n \to a \vdash M_i : \sigma_i \text{ for } i = 1 \ldots n}{\Gamma, x : \sigma_1 \to \ldots \to \sigma_n \to a \vdash x\ M_1 \ldots M_n : a}\ (\to_L \mathrm{E})$$

Clearly, longness is not violated by generalization (cf. Lemma 7) and η-expansion does not violate principality (cf. Lemma 8).

▶ **Lemma 7.** *If $M \in \mathrm{Long}(S(\tau))$ and $\vdash M : \tau$, then $M \in \mathrm{Long}(\tau)$.*

▶ **Lemma 8** ([8, 8A11.2])**.** *If a β-normal form M has the principal type τ, then its unique η-expansion $M^+ \in \mathrm{Long}(\tau)$ has the principal type τ.*

Our main result is that the following principal inhabitation problem (cf. Problem 1) is PSPACE-complete.

▶ **Problem 1** (Principal Inhabitation)**.** *Given a simple type τ, is there a λ-term M in β-normal form such that τ is the principal type of M?*

▶ **Theorem 9.** *The principal inhabitation problem (cf. Problem 1) is* PSPACE*-complete.*

Proof. The upper bound is shown in Section 5 Lemma 38 and the lower bound is shown in Section 6 Lemma 42. ◀

3 Subformula Filtration

The *subformula filtration* technique, which was developed for the intersection type system in [7], eliminates unnecessary structure in type derivations. It can be used to show that if M is typable, then there exists a type derivation $\mathcal{D}$ for M such that any subformula of any type occurring in $\mathcal{D}$ also appears on the right-hand side of some judgement in $\mathcal{D}$. This can be seen as a generalization of the standard subformula property that only requires right-hand sides of judgements in $\mathcal{D}$ to be subformulae of types appearing in the root judgement of $\mathcal{D}$. Transferring the technique to the simply typed λ-calculus we obtain a necessary condition for principal inhabitation (cf. Lemma 14).

First, we adapt definitions from [7] to the simply typed λ-calculus, including that of the set $T(\mathcal{D})$ of types occurring on the right-hand sides of judgements in a given type derivation $\mathcal{D}$, and that of the notion of type filtration.

▶ **Definition 10** ($T(\mathcal{D})$)**.** Given a type derivation $\mathcal{D}$ we define the set $T(\mathcal{D}) = \{\tau \mid \Gamma \vdash M : \tau \text{ is a judgement in } \mathcal{D}\}$.

▶ **Definition 11** (Filtration Function $\mathcal{F}^a_X$)**.** Given a set X of types and a type atom a we define the *filtration function* $\mathcal{F}^a_X$ as follows

$$\mathcal{F}^a_X(b) = a \qquad \mathcal{F}^a_X(\sigma \to \tau) = \begin{cases} \mathcal{F}^a_X(\sigma) \to \mathcal{F}^a_X(\tau) & \text{if } \sigma \to \tau \in X \text{ and } \tau \in X \\ a & \text{otherwise} \end{cases}$$

Intuitively, a filtration function $\mathcal{F}^a_X$ collapses all type atoms and unnecessary subformulae wrt. X into a single type atom a. Let us tacitly lift filtration functions pointwise to type environments by $\mathcal{F}^a_X(\Gamma) = \{x : \mathcal{F}^a_X(\sigma) \mid (x : \sigma) \in \Gamma\}$.

Next, we formulate the corresponding filtration lemma for the simply typed λ-calculus.

▶ **Lemma 12.** *If $\mathcal{D} \triangleright \Gamma \vdash M : \tau$ and $T(\mathcal{D}) \subseteq X$, then $\mathcal{F}^a_X(\Gamma) \vdash M : \mathcal{F}^a_X(\tau)$, where a is fresh.*

Proof. Routine induction on the derivation $\mathcal{D}$.

Case (Ax): Clearly, $\mathcal{F}^a_X(\Gamma), x : \mathcal{F}^a_X(\sigma) \vdash x : \mathcal{F}^a_X(\sigma)$.

Case ($\to$I): The last rule is $\dfrac{\Gamma, x : \sigma' \vdash N : \tau'}{\Gamma \vdash \lambda x.M : \sigma' \to \tau'}\,(\to\mathrm{I})$.

We have $\tau' \in T(\mathcal{D}) \subseteq X$ and $\sigma' \to \tau' \in T(\mathcal{D}) \subseteq X$, therefore $\mathcal{F}^a_X(\sigma' \to \tau') = \mathcal{F}^a_X(\sigma') \to \mathcal{F}^a_X(\tau')$. By the induction hypothesis we have $\mathcal{F}^a_X(\Gamma), x : \mathcal{F}^a_X(\sigma') \vdash N : \mathcal{F}^a_X(\tau')$, which using ($\to$I) shows the claim.

Case ($\to$E): The last rule is $\dfrac{\Gamma \vdash N : \sigma' \to \tau' \qquad \Gamma \vdash L : \sigma'}{\Gamma \vdash N\ L : \tau'}\,(\to\mathrm{E})$.

We have $\tau' \in T(\mathcal{D}) \subseteq X$ and $\sigma' \to \tau' \in T(\mathcal{D}) \subseteq X$. Similarly to the previous case, the claim follows using the definition of $\mathcal{F}^a_X$, the induction hypothesis and the rule ($\to$E). ◀

The above Lemma 12 is useful to eliminate unnecessary subformulae in derivations as illustrated by the following Example 13.

▶ **Example 13.** Let $\sigma = b \to b$ and consider the derivation $\mathcal{D} = \dfrac{\dfrac{}{x : \sigma \vdash x : \sigma}\,(\mathrm{Ax})}{\vdash \lambda x.x : \sigma \to \sigma}\,(\to\mathrm{I})$.
We have $b \notin T(\mathcal{D}) = \{\sigma, \sigma \to \sigma\}$. Therefore, $\mathcal{D}$ contains unnecessary structure in order to type $\lambda x.x$. Applying $\mathcal{F}^a_{T(\mathcal{D})}$ we obtain $\dfrac{\dfrac{}{x : a \vdash x : a}\,(\mathrm{Ax})}{\vdash \lambda x.x : a \to a}\,(\to\mathrm{I})$, noting that $\mathcal{F}^a_{T(\mathcal{D})}(b \to b) = a$ because $b \notin T(\mathcal{D})$.

Finally, we conclude this section with a necessary condition for type derivations of principal types, which is connected to Property ($\star$) in Section 4 and, specifically, Lemma 30.

▶ **Lemma 14.** *If $\mathcal{D} \triangleright \emptyset \vdash M : \tau$ and τ contains a subformula $\sigma' \to \tau'$ such that $\tau' \notin T(\mathcal{D})$, then τ is not the principal type of M.*

Proof. By Lemma 12 we have $\emptyset \vdash M : \mathcal{F}^a_{T(\mathcal{D})}(\tau)$, where a is fresh. Since $\tau' \notin T(\mathcal{D})$, in $\mathcal{F}^a_{T(\mathcal{D})}(\tau)$ the corresponding subformula at the position of $\sigma' \to \tau'$ in τ is either undefined or a. Therefore, there is no substitution S such that $S(\tau) = \mathcal{F}^a_{T(\mathcal{D})}(\tau)$. ◀

4 Subformula Calculus

To distinguish distinct subformula occurrences in a given type τ, we use *paths* π in the syntax tree of τ, which are defined as follows

$$\pi \in \{1, 2\}^* .$$

Since paths are character sequences, we use abbreviations such as $\pi 2^n$ for the path π followed by n twos. We access a subformula at path π in a given type τ by $\tau(\pi)$, defined as

$$\tau(\varepsilon) = \tau\,, \qquad (\sigma \to \tau)(1\pi) = \sigma(\pi)\,, \qquad (\sigma \to \tau)(2\pi) = \tau(\pi)\,.$$

The above definition implies that we use types as functions from the set of their paths to their subformulae. In particular, $\mathrm{dom}(\tau)$ is the set of paths in τ and $\mathrm{ran}(\tau)$ is the set of subformulae in τ.

Similarly to the simply typed system, we define *path environments* $\Delta = \{x_1 : \pi_1, \dots, x_n : \pi_n\}$, where $\mathrm{dom}(\Delta) = \{x_1, \dots, x_n\}$. For a relation R on paths, the calculus $\vdash_R$ is given by rules $(\to_R\mathrm{I})$ and $(\to_R\mathrm{E})$ in the following Definition 15.

▶ **Definition 15** (Calculus $\vdash_R$)**.**

$$\frac{\Delta, x : \pi 1 \vdash_R M : \pi 2}{\Delta \vdash_R \lambda x.M : \pi}\,(\to_R\mathrm{I}) \qquad \frac{\pi 2^n \; R \; \pi' \qquad \Delta, x : \pi \vdash_R M_i : \pi 2^{i-1} 1 \text{ for } i = 1 \dots n}{\Delta, x : \pi \vdash_R x\, M_1 \dots M_n : \pi'}\,(\to_R\mathrm{E})$$

We call conditions of the form $\pi \; R \; \pi'$ *side conditions*. The above calculus $\vdash_R$, similarly to the calculus $\mathrm{TA}_{\mathrm{pln}}$ in [4], captures as side conditions identities imposed by the typed term. In contrast to $\mathrm{TA}_{\mathrm{pln}}$ it does not contain or require actual type information. Additionally, for any closed λ-term M in β-normal form there exists a relation R such that $\vdash_R M : \varepsilon$. Clearly, $\vdash_R$ is monotonous in the sense of the following Lemma 16.

▶ **Lemma 16.** *If $\vdash_R M : \varepsilon$ and $R \subseteq R'$, then $\vdash_{R'} M : \varepsilon$.*

As in the simply typed system, paths on the left-hand side (resp. right-hand side) of $\vdash_R$ are of negative (resp. positive) variance, which is formalized in the following Lemma 17.

▶ **Lemma 17.** *If $\mathcal{D} \triangleright \emptyset \vdash_R M : \varepsilon$, then each judgement $\Delta \vdash_R N : \pi$ in $\mathcal{D}$ satisfies*

(i) *The number of 1s in π is even.*

(ii) *For each $(x : \pi') \in \Delta$ the number of 1s in π' is odd.*

Proof. Induction on depth of derivation for the more general claim: if $\Delta \vdash_R M : \pi$ is derived by $\mathcal{D}$ and satisfies (i) and (ii), then each judgement in $\mathcal{D}$ satisfies (i) and (ii). Clearly, if the concluding judgement satisfies (i) and (ii), then all premise judgements satisfy (i) and (ii) in both $(\to_R\mathrm{I})$ and $(\to_R\mathrm{E})$. ◀

The above observation restricts paths in side conditions as follows.

▶ **Corollary 18.** *If $\mathcal{D} \triangleright \emptyset \vdash_R M : \varepsilon$, then $\mathcal{D}$ contains no side condition of the form $\pi \; R \; \pi$.*

Intuitively, a derivation in $\vdash_R$ contains (as side conditions) necessary equality constraints on atomic subformulae that are required for typing a given term M. Therefore, we are interested in the minimal relation R such that $\vdash_R M : \varepsilon$.

Given a relation R let us denote the *reflexive, symmetric, transitive closure* of R by $R^{\equiv}$. Clearly, if $\vdash_R M : \varepsilon$, then $\vdash_{R^{\equiv}} M : \varepsilon$.

▶ **Definition 19** (R_M)**.** Given a λ-term M in β-normal form, let R_M be the minimal (wrt. inclusion) equivalence relation such that $\vdash_{R_M} M : \varepsilon$.

Derivations in $\vdash_R$ are uniquely defined by the concluding judgement, therefore, the minimal relation R of necessary side conditions is uniquely defined as well. By monotonicity (cf. Lemma 16) we can take $R_M = R^{\equiv}$.

▶ **Example 20.** We have $R_{\lambda x.\lambda y.x} = \{(1, 22)\}^{\equiv}$ and $R_{\lambda x.\lambda y.y} = \{(21, 22)\}^{\equiv}$. Note that the domain of $R_{\lambda x.\lambda y.x}$ (resp. $R_{\lambda x.\lambda y.y}$) does not contain the path 21 (resp. 1) which would correspond to the type of y (resp. x).

Similar to the simply typed system, we can identify term variables in the path environment that are bound to same paths.

▶ **Lemma 21.** $\Delta, x : \pi, y : \pi \vdash_R M : \pi'$ *iff* $\Delta, x : \pi \vdash_R M[y := x] : \pi'$.

Proof. Induction on the derivation. The structure of both derivations is identical. ◀

The above Lemma 21 has a subtle implication regarding interchangeability of abstracted variables in a given term M referring to same paths without changing the corresponding relation R_M. This property will be crucial in the upper bound construction in Section 5 and is illustrated in the following Example 22.

▶ **Example 22.** Consider $M = \lambda f.f\ (\lambda x.f\ (\lambda y.y))$ and $M' = \lambda f.f\ (\lambda x.f\ (\lambda y.x))$. Both M and M' are normal principal inhabitants of $((a \to a) \to a) \to a$. Let $\Delta = \{f : 1, x : 111, y : 111\}$. The only difference between the derivation of $\vdash_{R_M} M : \varepsilon$ and a derivation of $\vdash_{R_{M'}} M' : \varepsilon$ is the leaf judgement. For the former it is $\Delta \vdash_{R_M} y : 112$ and for the latter $\Delta \vdash_{R_{M'}} x : 112$. By Lemma 21 we have $\Delta \vdash_{R_{M'}} y : 112$ and $\Delta \vdash_{R_M} x : 112$. Since the rest of the derivations is identical, we have $R_M = R_{M'}$.

The equivalence relation R_M intuitively captures equality constraints on atomic subformulae imposed by a given term M. Complementarily, given a type τ, we are interested in equality constraints on atomic subformulae satisfied by τ. To capture such constraints we define the equivalence relation R_τ in the following Definition 23.

▶ **Definition 23** (R_τ)**.** Given a type τ we define the equivalence relation R_τ on paths in $\mathrm{dom}(\tau)$ as $R_\tau = \{(\pi, \pi') \mid \pi \neq \pi' \wedge \tau(\pi) = \tau(\pi') \in \mathbb{A}\}^{\equiv}$.

Observe that the condition $\pi \neq \pi'$ in the definition of R_τ excludes singular occurrences of type atoms in τ from the domain of R_τ while the subsequent equivalence closure ensures reflexivity. This is illustrated in the following Example 24.

▶ **Example 24.** We have $R_{a\to b\to a} = \{(1, 22)\}^{\equiv}$ and $R_{a\to b\to b} = \{(21, 22)\}^{\equiv}$. Similarly to Example 20 the domain of $R_{a\to b\to a}$ (resp. $R_{a\to b\to b}$) does not contain the path 21 (resp. 1).

Due to structural similarity between the rules ($\to_L$E) and ($\to_R$E) we obtain a simple characterization of long normal inhabitants of a given type in the following Lemma 25.

▶ **Lemma 25.** *Given a λ-term M in β-normal form, the following conditions are equivalent*

(i) $M \in \mathrm{Long}(\tau)$,
(ii) $\vdash_{R_\tau} M : \varepsilon$,
(iii) $R_M \subseteq R_\tau$.

Proof.

(i) $\Longrightarrow$ **(ii):** Assume $\mathcal{D} \triangleright \emptyset \vdash M : \tau$ using only the rules ($\to$I) and ($\to_L$E) (cf. Definition 6). By routine induction on $\mathcal{D}' \triangleright \emptyset \vdash_{R_M} M : \varepsilon$ we have that for each judgement $\Delta \vdash_{R_M} N : \pi$ in $\mathcal{D}'$ there is a judgment $\{x : \tau(\pi') \mid (x : \pi') \in \Delta\} \vdash N : \tau(\pi)$ in $\mathcal{D}$. Therefore, if $\Delta, x : \pi \vdash_{R_M} x\ M_1 \dots M_n : \pi'$ is a judgement in $\mathcal{D}'$, then $\tau(\pi 2^n) = \tau(\pi') \in \mathbb{A}$. By Corollary 18 we additionally have $\pi 2^n \neq \pi'$, and ultimately $(\pi 2^n, \pi') \in R_\tau$. Therefore, $\vdash_{R_\tau} M : \varepsilon$.

(ii) $\Longrightarrow$ **(iii):** If $\vdash_{R_\tau} M : \varepsilon$ but $R_M \not\subseteq R_\tau$, then R_M is not minimal, which contradicts the definition of R_M.

(iii) $\Longrightarrow$ (i): Assume $R_M \subseteq R_\tau$. We directly translate the derivation of $\vdash_{R_M} M : \varepsilon$ to a derivation using rules ($\to$I) and ($\to_L$E) (cf. Definition 6). The side condition $\pi 2^n \; R_M \; \pi'$ in ($\to_{R_M}$E) implies $\tau(\pi 2^n) = \tau(\pi') \in \mathbb{A}$. Additionally, in case of ($\to_{R_M}$I) we have that $\pi 1 \in \mathrm{dom}(\tau)$ iff $\pi 2 \in \mathrm{dom}(\tau)$. ◀

Since R_M contains atomic equality constraints imposed by M, we are free to rename some atomic subformulae in a given type τ without violating type derivations wrt. M.

▶ **Lemma 26.** *Given a λ-term $M \in \mathrm{Long}(\tau)$ and $\pi \in \mathrm{dom}(\tau)$ such that $\tau(\pi) = a$. Let b be a fresh atom. Define τ' by τ replacing for each $\pi' \in \mathrm{dom}(\tau)$ such that $\pi = \pi'$ or $\pi \; R_M \; \pi'$ the subformula a at π' by b. Then $M \in \mathrm{Long}(\tau')$.*

Proof. $M \in \mathrm{Long}(\tau)$ by Lemma 25 implies $R_M \subseteq R_\tau$. Renaming subformulae a in τ at path π and at all paths π' with $\pi \; R_M \; \pi'$ to b preserves $R_M \subseteq R_{\tau'}$. By Lemma 25 we obtain $M \in \mathrm{Long}(\tau')$. ◀

Next, we formulate a necessary condition (cf. Lemma 27) for principal inhabitation.

▶ **Lemma 27.** *Given a type τ let $M \in \mathrm{Long}(\tau)$. If τ is the principal type of M, then $R_\tau = R_M$.*

Proof. Since $M \in \mathrm{Long}(\tau)$, by Lemma 25 we have $R_M \subseteq R_\tau$. Assume there exists $(\pi, \pi') \in R_\tau$ such that $(\pi, \pi') \notin R_M$. Let a be a fresh atom. Define τ' by renaming each subformula of τ in $\{\pi'' \mid \pi'' = \pi \text{ or } \pi'' \; R_M \; \pi\}$ to a. Since $\tau(\pi) \in \mathbb{A}$, by Lemma 26 we have $M \in \mathrm{Long}(\tau')$. However, τ' is strictly more general than τ. ◀

Unfortunately, the converse of the above Lemma 27 is not true as illustrated in the following Example 28.

▶ **Example 28.** Consider $M = \lambda x.\lambda y.x$ and $\tau = a \to (b \to c) \to a$. We have $R_M = \{(1, 22)\}^{\equiv} = R_\tau$. However, τ has no normal principal inhabitant.

One could follow the approach of [4] of marking necessary arrows in derivations (requiring further interplay between terms, derivations and types) to close the gap exposed in the above Example 28. At first sight, taking arrow subformulae in derivations into account appears inevitable. Surprisingly, this is not the case. As stated by Lemma 14 in Section 3, certain types (such as $a \to (b \to c) \to a$) have no normal principal inhabitants. Strikingly, formulated as a necessary (and easy to verify) condition ($\star$) in the following Definition 29 we are able to close the mentioned gap without additional constraints on terms or derivations.

▶ **Definition 29** (($\star$))**.** We say τ satisfies ($\star$), if $\forall \pi \in \mathrm{dom}(\tau).(\tau(\pi 2) \in \mathbb{A} \Rightarrow (\pi 2, \pi 2) \in R_\tau)$.

Intuitively, a given type τ satisfies ($\star$), if τ has no subformula $\sigma \to a$, where a occurs exactly once as a subformula of τ. This coincides with the first property in [5, Proposition 4.3] and is a necessary condition for principal inhabitation, as shown by the following Lemma 30.

▶ **Lemma 30.** *If τ does not satisfy ($\star$), then τ has no normal principal inhabitant.*

Proof. If τ does not satisfy ($\star$), then there exists a path $\pi \in \mathrm{dom}(\tau)$ such that $\tau(\pi 2) \in \mathbb{A}$ and $(\pi 2, \pi 2) \notin R_\tau$. Assume τ has a normal principal inhabitant $M \in \mathrm{Long}(\tau)$ (cf. Lemma 8). By Lemma 25 there exists a derivation $\mathcal{D} \triangleright \emptyset \vdash_{R_\tau} M : \varepsilon$. Since $(\pi 2, \pi 2) \notin R_\tau$ the derivation $\mathcal{D}$ contains no judgement of the shape $\Delta \vdash_{R_\tau} L : \pi 2$ for some path environment Δ and term L. Therefore, replacing paths by corresponding subformulae in τ, there exists a derivation $\mathcal{D}' \triangleright \emptyset \vdash M : \tau$ such that $a \notin T(\mathcal{D}')$, where $\tau(\pi) = \sigma \to a$ for some type σ. By Lemma 14 the type τ is not the principal type of M, which is a contradiction. ◀

Finally, we formulate a sufficient condition (cf. Lemma 31) for principal inhabitation.

▶ **Lemma 31.** *Given a type τ satisfying $(\star)$ let $M \in \mathrm{Long}(\tau)$. If $R_\tau = R_M$, then τ is the principal type of M.*

Proof. Assume M has a strictly more general principal type τ'. Fix the substitution S such that $S(\tau') = \tau$. By Lemma 7 we have $M \in \mathrm{Long}(\tau')$. Therefore, by Lemma 27 we have $R_M = R_{\tau'}$. We show that $R_\tau \neq R_{\tau'}$.

Case $S : \mathbb{A} \to \mathbb{A}$: There exist π, π' such that $\tau(\pi) = \tau(\pi') \in \mathbb{A}$ and $\tau'(\pi) \neq \tau'(\pi')$. Therefore, $(\pi, \pi') \in R_\tau$ but $(\pi, \pi') \notin R_{\tau'} = R_M$.

Case $S(a) = \sigma_1 \to \ldots \to \sigma_n \to b$ for some $n > 0$ and $a \in \mathrm{ran}(\tau') \cap \mathbb{A}$: Fix any path $\pi \in \mathrm{dom}(\tau')$ such that $\tau'(\pi) = a$. Since $\tau(\pi 2^n) = b$ and $n > 0$, due to $(\star)$ we have $(\pi 2^n, \pi 2^n) \in R_\tau$. However, $\tau'(\pi 2^n)$ is undefined, therefore $(\pi 2^n, \pi 2^n) \notin R_{\tau'} = R_M$. ◀

In sum, the equality $R_M = R_\tau$ characterizes principality in the sense of the following Theorem 32.

▶ **Theorem 32.** *Given a type τ satisfying $(\star)$ and a λ-term $M \in \mathrm{Long}(\tau)$ we have that τ is the principal type of M iff $R_M = R_\tau$.*

Proof. '$\Longrightarrow$' by Lemma 27. '$\Longleftarrow$' by Lemma 31. ◀

Bearing resemblance to the characterization in [4, Proposition 17], the above characterization in Theorem 32 has two benefits. First, it does not require marking of arrows in derivations. Second, it is factored into R_M (uniquely defined by M) and R_τ (uniquely defined by τ). Since by Lemma 25 any long normal inhabitant M of τ satisfies $R_M \subseteq R_\tau$ and the size of R_τ is polynomial in the size of τ, we will only require polynomial space for principal inhabitation in the following Section 5.

5 PSPACE Upper Bound

In this section we develop a polynomial space algorithm to decide principal inhabitation. As mentioned in the introduction, there are three hurdles to overcome to get a polynomial space upper bound.

First, if $\Gamma \vdash M : \tau$ is derivable in the simple type system, then there is a derivation of that judgement which does not contain any judgement $\Gamma \vdash M' : \tau$ such that $M \neq M'$. For principal inhabitation this does not hold as shown in the following Example 33. This issue is solved by taking into account the impact on R_M by corresponding judgements.

▶ **Example 33.** Let $\tau = (a \to a) \to a \to a$. The normal principal inhabitants of τ are exactly the Church numerals greater equal to two, i.e. $\lambda f.\lambda x.f\ (f\ x)$, $\lambda f.\lambda x.f\ (f\ (f\ x))$, ... The corresponding type derivations necessarily assign the type a to the terms $x, f\ x$ and $f\ (f\ x)$ in identical type environments.

Second, term variables with identical types are interchangeable in the simple type system. However, this may violate principality as shown in the following Example 34. This issue is solved using Lemma 21, due to which an identification of x and y is allowed, if x and y are both bound to the same subformula occurrence, i.e. the same path.

▶ **Example 34.** Let $\tau = (a \to a \to a) \to a \to a \to a$, $M = \lambda f.\lambda x.\lambda y.f\ (f\ x\ y)\ (f\ y\ x)$, $M_x = \lambda f.\lambda x.\lambda y.f\ (f\ x\ x)\ (f\ x\ x)$ and $M_y = \lambda f.\lambda x.\lambda y.f\ (f\ y\ y)\ (f\ y\ y)$. Each M, M_x and M_y is an inhabitant of τ. However, only M of the three is a normal principal inhabitant of τ.

Algorithm 1 Algorithm INH deciding existence of normal principal inhabitants

1: *Input:* simple type τ
2: *Output:* **accept** iff there exists a normal principal inhabitant of τ
3: **if** $\neg\Big(\forall \pi \in \mathrm{dom}(\tau).(\tau(\pi 2) \in \mathbb{A} \Rightarrow (\pi 2, \pi 2) \in R_\tau)\Big)$ **then**
4: **fail**
5: **end if**
6: $R := \mathsf{AUX}(\tau, \emptyset, \varepsilon, \emptyset)$
7: **if** $R = R_\tau$ **then**
8: **accept**
9: **else**
10: **fail**
11: **end if**

Algorithm 2 Non-deterministic Algorithm AUX

1: *Input:* simple type τ, set of paths P, path π, relation on paths R
2: *Output:* updated relation on paths R
3: **if** $\tau(\pi) = \sigma \to \tau$ **then**
4: **return** $\mathsf{AUX}(\tau, P \cup \{\pi 1\}, \pi 2, R)$
5: **else if** $\tau(\pi) = a$ for some a **then**
6: **choose** $\pi' \in P$ such that $\tau(\pi' 2^n) = a$ for some $n \geq 0$
7: $R := (R \cup \{(\pi' 2^n, \pi)\})^{\equiv}$
8: **for** $i = 1$ **to** n **do**
9: $R := \mathsf{AUX}(\tau, P, \pi' 2^{i-1} 1, R)$
10: **end for**
11: **end if**
12: **return** R

Third, a normal principal inhabitant M of a given type τ may be of exponential size. Therefore, we cannot in polynomial space construct an inhabitant explicitly and then check for principality as in [4, 5]. This issue is solved using the characterization in Theorem 32. Particularly, instead of M it suffices to construct R_M of size at most the size of R_τ, which is polynomial in the size of τ. This key observation allows us to stay in polynomial space.

Given a type τ, the idea behind the following Algorithm 1 to decide principal inhabitation (cf. Problem 1) is as follows. Start by verifying that τ satisfies $(\star)$. Continue with the auxiliary Algorithm AUX to construct a relation R corresponding to R_M for some long normal inhabitant M (which is not constructed explicitly). Last, verify $R_M = R_\tau$.

▶ **Example 35.** Let $\tau = ((a \to a) \to a) \to a$ and consider $\mathsf{INH}(\tau)$. Since τ satisfies $(\star)$, the condition in line 3 does not trigger a failure.

- Proceed with $\mathsf{AUX}(\tau, \emptyset, \varepsilon, \emptyset)$, which corresponds to inhabitant search of $\tau(\varepsilon) = \tau$.
- Since $\tau(\varepsilon)$ is an arrow type, take the first branch (line 4). This induces a potential inhabitant to be of the shape $\lambda f.N$ for a fresh f and some λ-term N. Proceed with $\mathsf{AUX}(\tau, \{1\}, 2, \emptyset)$, which corresponds to the search for N of type $\tau(2) = a$ in the type environment $\{f : \tau(1) = (a \to a) \to a\}$.
- Since $\tau(2) = a = \tau(12)$, take the second branch (lines 6–10) choosing the path $1 \in P$. This induces $N = f\ L$ for some λ-term L. Proceed with $\mathsf{AUX}(\tau, \{1\}, 11, \{(12, 2)\}^{\equiv})$, searching for L of type $\tau(11) = a \to a$ in the type environment $\{f : \tau(1) = (a \to a) \to a\}$.
- Since $\tau(11) = a \to a$ is an arrow type, take the first branch, i.e. $L = \lambda x.L'$ for a fresh x

and some λ-term L'. Proceed with $\mathsf{AUX}(\tau, \{1, 111\}, 112, \{(12, 2)\}^{\equiv})$, searching for L' of type $\tau(112) = a$ in the type environment $\{f : \tau(1) = (a \to a) \to a, x : \tau(111) = a\}$.

- Since $\tau(112) = a$, take the second branch. There are two options. The first option is to choose the path 111, since $\tau(112) = \tau(111)$. In this case, AUX would return control to INH with the result $R = \{(12, 2), (111, 112)\}^{\equiv}$ and INH would fail. The corresponding run of INH would induce the inhabitant $\lambda f.f\ (\lambda x.x)$, which is not a normal principal inhabitant of τ. The second option is to choose the path 1 since $\tau(112) = \tau(12)$ and proceed with $\mathsf{AUX}(\tau, \{1, 111\}, 11, \{(12, 2), (12, 112)\}^{\equiv})$. Choose the second option.
- Again, $\tau(11) = a \to a$ is an arrow type, take the first branch and proceed with $\mathsf{AUX}(\tau, \{1, 111\}, 112, \{(12, 2), (12, 112)\}^{\equiv})$.
- Again, $\tau(112) = a$, take the second branch, choosing the path 111. After AUX returns $R = \{(12, 2), (12, 112), (111, 112)\}^{\equiv}$ to INH, INH accepts. The corresponding run of INH induces the normal principal inhabitant $\lambda f.f\ (\lambda x.f\ (\lambda y.x))$ (cf. Example 22) of τ.

▶ **Lemma 36** (Soundness of INH). *Given a type τ, if Algorithm 1 accepts, then there exists a normal principal inhabitant of τ.*

Proof. A successful run of Algorithm 1 induces a type derivation $\mathcal{D} \triangleright \emptyset \vdash_{R_M} M : \varepsilon$ for some M. In particular, line 4 in Algorithm AUX induces a λ-abstraction and lines 6–10 in Algorithm AUX induce an application with head variable of type $\tau(\pi')$ and n arguments. By Lemma 21 it suffices to take the variable that is bound to π' and in M is abstracted outermost. Line 3 in in Algorithm INH ensures that τ satisfies $(\star)$ and line 7 ensures that $R_M = R_\tau$. By Theorem 32 the term M is a normal principal inhabitant of τ. ◀

▶ **Lemma 37** (Completeness of INH). *Given a type τ, if there exists a normal principal inhabitant of τ, then there exists an accepting run of Algorithm 1 requiring at most polynomial space in the size of τ.*

Proof. Assume that τ has a normal principal inhabitant M. By Theorem 32 we have that τ satisfies Property $(\star)$ and there exists a normal principal inhabitant $M' \in \mathrm{Long}(\tau)$ such that $\mathcal{D} \triangleright \emptyset \vdash_{R_{M'}} M' : \varepsilon$ and $R_{M'} = R_\tau$. By induction on $\mathcal{D}$ there exists an accepting run $\mathcal{R}$ of Algorithm INH such that for each judgement $\Delta \vdash_{R_{M'}} L : \pi$ in $\mathcal{D}$ the run $\mathcal{R}$ invokes $\mathsf{AUX}(\tau, \mathrm{ran}(\Delta), \pi, R)$ where $R \subseteq R_{M'}$. Therefore, for each side condition $\pi'\ R_{M'}\ \pi''$ in $\mathcal{D}$ the corresponding invocation of AUX in line 7 ensures $\pi'\ R\ \pi''$. Overall, by Theorem 32 we have $R_\tau = R_{M'} = R$ and INH accepts.

Space requirements: The parameters τ, $P \subseteq \mathrm{dom}(\tau)$, $\pi \in \mathrm{dom}(\tau)$ and $R \subseteq \mathrm{dom}(\tau)^2$ are polynomial in the size of τ. Since the above run $\mathcal{R}$ is accepting and there are no side-effects, there exists an accepting run $\mathcal{R}'$ that has no invocations with identical parameters along the recursion branches of AUX. Since P and R are non-decreasing along the recursion branches of AUX, the invocation stack of AUX in $\mathcal{R}'$ is of polynomial depth in size of τ. ◀

▶ **Lemma 38.** *Problem 1 is in* PSPACE.

Proof. By Lemma 36, Lemma 37 and the identity PSPACE=NPSPACE. ◀

6 PSPACE Lower Bound

In this section we establish a PSPACE lower bound for principal inhabitation. Unfortunately, the standard reduction (cf. [12]) from quantified Boolean formulae to inhabitation in the simply typed λ-calculus does not carry over immediately as illustrated by the following Example 39.

▶ **Example 39.** Consider the formula $\varphi = \exists p.\psi$, where $\psi = p \vee \neg p$. By the construction in [12] φ is true iff the type $\sigma = ((a_p \to a_\psi) \to a_\varphi) \to ((a_{\neg p} \to a_\psi) \to a_\varphi) \to (a_p \to a_\psi) \to (a_{\neg p} \to a_\psi) \to a_\varphi$ is inhabited in the simply typed λ-calculus. The only long normal inhabitants of σ are $\lambda x_1.\lambda x_2.\lambda y_1.\lambda y_2.x_1\ (\lambda z.y_1\ z)$ and $\lambda x_1.\lambda x_2.\lambda y_1.\lambda y_2.x_2\ (\lambda z.y_2\ z)$ for both of which σ is not principal. Although φ is true, there is no normal principal inhabitant of σ.

The inherent issue with the standard approach is that existential quantifiers and disjunctions may introduce unnecessary (or even unusable) subformulae. We solve this issue by introducing additional subformulae not affecting inhabitability to secure principal inhabitability.

The construction in [12] shows that the following Problem 2, which is a restriction of inhabitation in the simply typed λ-calculus, is PSPACE-hard.

▶ **Problem 2.** *Given a type $\tau = \sigma_1 \to \ldots \to \sigma_n \to a$ such that $\sigma_i = (b_i^1 \to c_i^1) \to (b_i^2 \to c_i^2) \to d_i$ for some $b_i^1, c_i^1, b_i^2, c_i^2, d_i \in \mathbb{A}$ for $i = 1 \ldots n$, is there a λ-term M such that $\vdash M : \tau$?*

Note that the exact construction in [12] also uses types of the shape $a \to b$ which can be represented by $(c \to a) \to (c \to a) \to b$ where c is fresh.

In the remainder of this section we fix a simple type τ according to Problem 2 with corresponding subformulae $\sigma_1, \ldots, \sigma_n$ and a. Our goal is to construct a type τ^* such that τ is inhabited iff τ^* is principally inhabited. Let $\{a_1, \ldots, a_l\}$ be the set of type atoms in τ and fix k such that $a = a_k$. We construct τ^* (of size polynomial in the size of τ) as follows

$$\begin{aligned}\tau^* =&((a_1 \to \ldots \to a_l \to a) \to a \to a) \to (a_1 \to a_1 \to a_1) \to \ldots \to (a_1 \to a_l \to a_l)\\ &\to (a_2 \to a_1 \to a_1) \to \ldots \to (a_2 \to a_l \to a_l)\\ &\to \ldots \to (a_l \to a_1 \to a_1) \to \ldots \to (a_l \to a_l \to a_l)\\ &\to (a \to a) \to \sigma_1 \to \ldots \to \sigma_n \to a\end{aligned}$$

Since all additional arguments in τ^* are intuitionistically valid formulae, an inhabitant of τ^* induces an inhabitant of τ.

▶ **Lemma 40.** *If $\vdash M : \tau^*$, then $\vdash M \underbrace{K^* \ldots K^*}_{1+l^2 \text{ times}} I : \tau$, where $I = \lambda x.x$ and $K^* = \lambda x.\lambda y.y$.*

It remains to show that if τ is inhabited, then τ^* is principally inhabited.

▶ **Lemma 41.** *If τ has an inhabitant, then τ^* has a normal principal inhabitant.*

Proof. Assume that τ has an inhabitant, then there exists a λ-term N such that $\{w_1 : \sigma_1, \ldots, w_n : \sigma_n\} \vdash N : a$ and N is in long β-normal form. Define the λ-term M^* as follows

$$\begin{aligned}&M^* = \lambda z.\lambda x_1^1 \ldots \lambda x_1^l.\lambda x_2^1 \ldots \lambda x_2^l \ldots \lambda x_l^1 \ldots \lambda x_l^l.\lambda x.\lambda w_1 \ldots \lambda w_n.x\ (z\ F\ (x\ N))\\ &F = \lambda y_1 \ldots \lambda y_l.x_1^k\ G_1^1\ (x\ (x\ y_k))\\ &G_i^j = x_{j+1}^j\ G_i^{j+1}\ (x_i^j\ y_i\ (x_i^j\ y_i\ y_j)) \text{ for } i = 1 \ldots l, j = 1 \ldots l-1\\ &G_i^l = x_1^l\ G_{i+1}^1\ (x_i^l\ y_i\ (x_i^l\ y_i\ y_l)) \text{ for } i = 1 \ldots l-1\\ &G_l^l = x_1^l\ H_1\ (x_l^l\ y_i\ (x_l^l\ y_l\ y_l)) \text{ for } i = 1 \ldots l-1\\ &H_i = x_j^i\ (w_i\ L_{i_1}^{i_2}\ L_{i_3}^{i_4})\ (x_{i+1}^i\ H_{i+1}\ y_i) \text{ for } i = 1 \ldots l-1\\ &\text{ where } \sigma_i = (a_{i_1} \to a_{i_2}) \to (a_{i_3} \to a_{i_4}) \to a_j\\ &H_l = x_j^l\ (w_l\ L_{i_1}^{i_2}\ L_{i_3}^{i_4})\ y_l \text{ where } \sigma_l = (a_{i_1} \to a_{i_2}) \to (a_{i_3} \to a_{i_4}) \to a_j\\ &L_i^j = \lambda t.x_i^j\ t\ y_j \text{ for } i = 1 \ldots l, j = 1 \ldots l\end{aligned}$$

We have $M^* \in \mathrm{Long}(\tau^*)$. Types of key subterms are outlined in the following overview.

$x : a_k \to a_k$	$x_i^j : a_i \to a_j \to a_j$ for $i = 1 \dots l, j = 1 \dots l$
$y_i : a_i$ for $i = 1 \dots l$	$z : (a_1 \to \dots \to a_l \to a_k) \to a_k \to a_k$
$w_i : \sigma_i$ for $i = 1 \dots n$	
$F : a_1 \to \dots \to a_l \to a$	$G_i^j : a_j$ for $i = 1 \dots l, j = 1 \dots l$
$H_i : a_i$ for $i = 1 \dots l$	$L_i^j : a_i \to a_j$

We use Theorem 32 to show that τ^* is the principal type of M^* by showing $R_{M^*} = R_{\tau^*}$. Since $M^* \in \mathrm{Long}(\tau^*)$, by Lemma 25 we have $R_{M^*} \subseteq R_{\tau^*}$. To obtain $R_{M^*} \supseteq R_{\tau^*}$, we show that for each path $\pi \in \mathrm{dom}(\tau^*)$ if $\tau^*(\pi) = a_i$, then $(\pi, 112^{i-1}1) \in R_{M^*}$. Therefore, if $(\pi, \pi') \in R_{\tau^*}$, then $\pi \; R_{M^*} \; 112^{i-1}1 \; R_{M^*} \; \pi'$ and $(\pi, \pi') \in R_{M^*}$ by transitivity. Let $\mathcal{D}$ be the derivation of $\vdash_{R_{M^*}} R_{M^*} : \varepsilon$.

Let $\pi_i^j = 22^{l\cdot(i-1)+(j-1)}1$ for $i, j = 1 \dots l$. We have $\tau^*(\pi_i^j) = a_i \to a_j \to a_j$. Let $\bar{\pi}_i = 112^{i-1}1$ for $i = 1 \dots l$. We have $\tau^*(\bar{\pi}_i) = a_i$. Let $\hat{\pi}_i = 2^{1+l^2+i}1$ for $i = 1 \dots n$. We have $\tau^*(\hat{\pi}_i) = \sigma_i$. In $\mathcal{D}$ the paths π_i^j are assigned to x_i^j, the paths $\bar{\pi}_i$ are assigned to y_i and the paths $\hat{\pi}_i$ are assigned to w_i.

For each $i, j = 1 \dots l$ the term M^* contains the subterm G_i^j. Leaving out some details by $[\dots]$, $\mathcal{D}$ contains the judgement $[\dots], x_i^j : \pi_i^j, y_i : \bar{\pi}_i, y_j : \bar{\pi}_j \vdash_{R_{M^*}} x_i^j \; y_i \; (x_i^j \; y_i \; y_j) : [\dots]$. The corresponding subderivation therefore entails $(\pi_i^j 1, \bar{\pi}_i) \in R_{M^*}$, $(\pi_i^j 21, \pi_i^j 22) \in R_{M^*}$ and $(\pi_i^j 21, \bar{\pi}_j) \in R_{M^*}$.

We proceed similarly with the remaining subformulae of τ^*. The most crucial subformulae are at paths $\hat{\pi}_i$ that correspond to σ_i for $i = 1 \dots n$. For each $i = 1 \dots n$ the term M^* contains the subterm H_i. Consider $\sigma_i = (a_{i_1} \to a_{i_2}) \to (a_{i_3} \to a_{i_4}) \to a_j$. Due to the subterm $x_j^i \; (w_i \; L_{i_1}^{i_2} \; L_{i_3}^{i_4}) \; [\dots]$ the corresponding subderivation entails $(\pi_j^i 1, \hat{\pi}_i 22) \in R_{M^*}$, therefore $(\bar{\pi}_j, \hat{\pi}_i 22) \in R_{M^*}$. Due to the subterm $w_i \; L_{i_1}^{i_2} \; L_{i_3}^{i_4}$ the corresponding subderivation entails $(\hat{\pi}_i 12, \pi_{i_1}^{i_2} 22) \in R_{M^*}$, $(\hat{\pi}_i 11, \pi_{i_1}^{i_2} 1) \in R_{M^*}$, $(\hat{\pi}_i 212, \pi_{i_3}^{i_4} 22) \in R_{M^*}$ and $(\hat{\pi}_i 211, \pi_{i_3}^{i_4} 1) \in R_{M^*}$. ◀

▶ **Lemma 42.** *Problem 1 is* PSPACE*-hard.*

Proof. By reduction from Problem 2 using Lemma 41 and Lemma 40. ◀

Finally, we conjecture that the construction in the proof of Lemma 41 can be generalized to arbitrary simple types (not restricted to the shape in Problem 2). The main idea is, instead of using the subformula $(a_1 \to \dots \to a_l \to a) \to a \to a$, to use the subformula $(\rho_1 \to \dots \rho_m \to a) \to a \to a$, where $\{\rho_1, \dots, \rho_m\}$ is the set of subformulae in the given type. Although the conjectured generalization is not necessary for the lower bound proof, it may be of systematic interest as a 'principal closure' of simple types.

7 Conclusion and Future Work

We have studied the problem of principal inhabitation in the simply typed λ-calculus, showing that the problem is PSPACE-complete. We believe that the techniques employed here (including filtration and path relations) condense the algorithmic essence of the problem. The presented polynomial space bounded algorithm should be a good starting point for further algorithm engineering for efficiency, relying on the subformula calculus and the logic of path relations it gives rise to.

In future work we intend to apply the algorithm in the context of type-based and combinatory logic synthesis [9, 6, 3]. In this context, we plan to add a facility for synthesizing normal principal inhabitants as combinators of general applicability in component repositories. Further, when types and corresponding terms are inductively defined, the provided

characterization of normal principal inhabitants may prove useful in mechanized certification of principality by proof assistants. Finally, the presented approach could be useful to inspect principal inhabitation in the simply typed λI-calculus for which inhabitation is 2-EXPTIME-complete [10].

References

1. H. P. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types*. Perspectives in Logic, Cambridge University Press, 2013.
2. C.-B. Ben-Yelles. *Type-assignment in the lambda-calculus; syntax and semantics*. PhD thesis, Mathematics Dept., University of Wales Swansea, UK, 1979.
3. Jan Bessai, Andrej Dudenhefner, Boris Düdder, Moritz Martens, and Jakob Rehof. Combinatory Logic Synthesizer. In *Leveraging Applications of Formal Methods, Verification and Validation, 6th International Symposium ISoLA 2014, Corfu, Greece, October 8-11, 2014*, pages 26–40, 2014. doi:10.1007/978-3-662-45234-9_3.
4. Sabine Broda and Luís Damas. Counting a Type's Principal Inhabitants. In *TLCA'99*, pages 69–82, 1999. doi:10.1007/3-540-48959-2_7.
5. Sabine Broda and Luís Damas. On long normal inhabitants of a type. *J. Log. Comput.*, 15(3):353–390, 2005. doi:10.1093/logcom/exi016.
6. Boris Düdder, Moritz Martens, and Jakob Rehof. Staged Composition Synthesis. In *ESOP 2014, Proceedings*, pages 67–86, 2014. doi:10.1007/978-3-642-54833-8_5.
7. Andrej Dudenhefner and Jakob Rehof. Typability in Bounded Dimension. In *LICS 2017, Proceedings of the 32nd ACM/IEEE Symposium on Logic in Computer Science, Reykjavik, Iceland, June*, 2017.
8. J. Roger Hindley. *Basic Simple Type Theory*. Cambridge Tracts in Theoretical Computer Science, vol. 42, Cambridge University Press, 2008.
9. Jakob Rehof. Towards Combinatory Logic Synthesis. In *BEAT 2013, 1st International Workshop on Behavioural Types*. ACM, 2013.
10. Sylvain Schmitz. Implicational relevance logic is 2-exptime-complete. *J. Symb. Log.*, 81(2):641–661, 2016. doi:10.1017/jsl.2015.7.
11. Richard Statman. Intuitionistic Propositional Logic is Polynomial-space Complete. *Theoretical Computer Science*, 9:67–72, 1979. doi:10.1016/0304-3975(79)90006-9.
12. P. Urzyczyn. Inhabitation in Typed Lambda-Calculi (A Syntactic Approach). In *TLCA'97, Typed Lambda Calculi and Applications, Proceedings*, volume 1210 of *LNCS*, pages 373–389. Springer, 1997. doi:10.1007/3-540-62688-3-47.

List Objects with Algebraic Structure

Marcelo Fiore[1] **and Philip Saville**[2]

1 **Computer Laboratory, University of Cambridge, Cambridge, UK**
`Marcelo.Fiore@cl.cam.ac.uk`
2 **Computer Laboratory, University of Cambridge, Cambridge, UK**
`Philip.Saville@cl.cam.ac.uk`

Abstract

We introduce and study the notion of list object with algebraic structure. The first key aspect of our development is that the notion of list object is considered in the context of monoidal structure; the second key aspect is that we further equip list objects with algebraic structure in this setting. Within our framework, we observe that list objects give rise to free monoids and moreover show that this remains so in the presence of algebraic structure. In addition, we provide a basic theory explicitly describing as an inductively defined object such free monoids with suitably compatible algebraic structure in common practical situations. This theory is accompanied by the study of two technical themes that, besides being of interest in their own right, are important for establishing applications. These themes are: parametrised initiality, central to the universal property defining list objects; and approaches to algebraic structure, in particular in the context of monoidal theories. The latter leads naturally to a notion of nsr (or near semiring) category of independent interest. With the theoretical development in place, we touch upon a variety of applications, considering Natural Numbers Objects in domain theory, giving a universal property for the monadic list transformer, providing free instances of algebraic extensions of the Haskell Monad type class, elucidating the algebraic character of the construction of opetopes in higher-dimensional algebra, and considering free models of second-order algebraic theories.

1998 ACM Subject Classification D.1.1 Applicative (Functional) Programming, D.3.1 Formal Definitions and Theory, D.3.3 Language Constructs and Features, F.3.2 Semantics of Programming Languages, F.3.3 Studies of Program Constructs

Keywords and phrases list object, free monoid, strong monad, (cartesian, linear, and second-order) algebraic theory, near semiring, Haskell Monad type class, opetope

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.16

1 Introduction

Lists are a basic and fundamental construction in computer science and mathematics.

In type theory [35] and programming theory [37], the polymorphic type of lists $\mathrm{L}(X)$ is an inductive type with constructors

$$nil : \mathrm{L}(X)\,, \quad cons : X \times \mathrm{L}(X) \to \mathrm{L}(X)$$

and an eliminator supporting definitions by primitive recursion. Amongst the variety of possible primitive-recursion schemes, here we will be concerned with *pure iteration* [43]; by which from $n : P \to L$ and $c : X \times L \to L$ one may form the parametrised iterator $\mathrm{it}(n, c) : \mathrm{L}(X) \times P \to L$ defined by

$$\mathrm{it}(n,c)(nil, p) = n(p)\,, \quad \mathrm{it}(n,c)(cons(x,l), p) = c\big(x, \mathrm{it}(n,c)(l,p)\big)\,.$$

The type of Natural Numbers is then that of lists on the unit type.

In category theory, the list type is universally modelled by the notion of (parametrised) list object, directly generalising Lawvere's notion of NNO (Natural Numbers Object) [29]. A *list object* $\mathrm{L}(X)$ on an object X in a category with finite products $(1, \times)$, is an initial algebraic structure of the form

$$1 \xrightarrow{nil} \mathrm{L}(X) \xleftarrow{cons} X \times \mathrm{L}(X)\,.$$

Such a structure is a *parametrised list object* whenever it is parametrised initial, in the sense that for every structure $(P \xrightarrow{n} L \xleftarrow{c} X \times L)$ there exists a unique mediating map $\mathrm{it}(n, c) : \mathrm{L}(X) \times P \to L$ such that

$$\begin{array}{ccccc}
1 \times P & \xrightarrow{nil \times P} & \mathrm{L}(X) \times P & \xleftarrow{cons \times P} & X \times \mathrm{L}(X) \times P \\
\cong \downarrow & & \downarrow \mathrm{it}(n,c) & & \downarrow X \times \mathrm{it}(n,c) \\
P & \xrightarrow[n]{} & L & \xleftarrow[c]{} & X \times L
\end{array}$$

It is well-known that when the binary product $(\times)$ is closed these notions of list object coincide, and that in the further presence of binary coproducts $(+)$ one may explicitly describe list objects by means of initial algebras:

$$\mathrm{L}(X) = \mu A.\, 1 + X \times A$$

where we use the common notation $\mu A.\, F(A)$ for the initial F-algebra. However, there are important scenarios (*e.g.* pretoposes [21, 33]) where the product structure is not closed. For them the more general parametrised concept is the appropriate one, and we henceforth adopt it as primitive.

In this paper, we investigate two orthogonal generalisations of list objects. One generalisation is concerned with weakening the cartesian structure of list objects to a monoidal one (Definition 3.1). Whilst NNOs have been studied in cartesian [29] and monoidal [6, 38] settings, list objects have been mainly considered in the cartesian one. The other generalisation is novel in that we equip list objects with further algebraic structure with respect to which the iterator is an algebra homomorphism (Definition 5.1).

Section 2 provides the necessary categorical background for setting up the above definitions. List objects in monoidal categories are introduced in Section 3, where we observe that they give rise to free monoids (Lemma 3.4). A main example is the monad freely generated by an endofunctor (Example 3.5). The parametrisation aspect inherent to list objects is considered and studied in broader generality in Section 4. In particular, we present a theory that accounts for the parametrised initiality of list objects as iteratively-constructed initial algebras

$$\mathrm{L}(X) = \mu A.\, I + X \otimes A$$

in contexts where the monoidal structure $(I, \otimes)$ may not be left-closed (Corollary 4.9). This general theory applies in the further context of list objects with algebraic structure (specifically Lemma 4.8) and plays an essential role in the application to higher-dimensional algebra that we present in Section 7.2.2.

The central notion of list object with T-algebraic structure, as prescribed by a strong monad T, is introduced in Section 5 and referred to as T-list object (Definition 5.1). This is accompanied by a related notion of monoid with compatible T-algebraic structure, referred

to as T-monoid (Definition 5.2). These two notions are related as follows: firstly, we observe that T-list objects $\mathrm{M}(X)$ yield free T-monoids (Lemma 5.6); secondly, we give an explicit description of T-list objects as parametrised initial algebras (Lemma 5.7):

$$\mathrm{M}_T(X) = \mu A.\, T(I + X \otimes A)$$

that is available in common practical situations (Corollaries 5.10 and 5.11). Overall, thus, this yields a universal inductive description of free T-monoids by means of parametrised initial-algebras that is of wide applicability (Theorem 5.8). This result, which we had obtained independently, also features in the recent work of Piróg [39], who establishes it by different methods.

Throughout the above development, algebraic structure is considered abstractly as encapsulated by the notion of strong monad. Section 6 complements this from the perspective of algebraic theories, whilst Section 7 discusses applications to programming semantics and algebraic theories presented in the context of related work. The study of linear algebraic theories (Section 6.2) led us to the notion of nsr (or near semiring) category (Definition 6.2), which, besides being of independent interest, plays a role in the applications to functional programming and higher-dimensional algebra respectively presented in Sections 7.2.1 and 7.2.2. Finally, Section 8 concludes with a brief discussion on the significance of our results, and indications of ongoing work.

2 Algebraic structure in monoidal categories

We begin by outlining the basic categorical notions of algebraic and monoidal structure which are relevant to our development.

Strong monoidal structures. Throughout we work with *monoidal categories* $(\mathcal{C}, I, \otimes)$. To simplify notation, we replace the structural isomorphisms by an unlabelled $\cong$ in diagrams.

We will be considering *strong multiary endofunctors* (F, st) on monoidal categories $(\mathcal{C}, I, \otimes)$ consisting of a functor $F : \mathcal{C}^n \to \mathcal{C}$ $(n \in \mathbb{N})$ together with a *strength* st, which is given by a natural transformation $st_{(X_1,\ldots,X_n),Y} : F(X_1, \ldots, X_n) \otimes Y \to F(X_1 \otimes Y, \ldots X_n \otimes Y)$ subject to coherence conditions. Maps of strong endofunctors $(F, st) \to (F', st')$ are *strong natural transformations* given by natural transformations $F \to F'$ satisfying coherence conditions. (See *e.g.* [25] for details, or [10, Section I.1.2] for related, more general, notions.)

A *strong monad* $T = (T, st, \mu, \eta)$ is a strong functor (T, st) equipped with strong natural transformations $\eta : (\mathrm{Id}, \mathrm{id}) \to (T, st)$ and $\mu : (TT, T(st) \circ st) \to (T, st)$ subject to the monad laws. (See *e.g.* [32, Section VI.1].)

We will be specifically concerned with two kinds of algebraic structure on objects in a monoidal category: monoids and monad algebras. We recall both notions.

Monoids. A *monoid* (M, e, m) in a monoidal category $(\mathcal{C}, I, \otimes)$ consists of an object $M \in \mathcal{C}$ together with a *unit* map $e : I \to M$ and a *multiplication* map $m : M \otimes M \to M$, subject to identity and associativity laws. *Homomorphisms* of monoids are morphisms of the underlying objects that preserve units and commute with the multiplications. We write $Mon(\mathcal{C})$ for the corresponding category. (See *e.g.* [32, Section VII.3] for details.)

Monad algebras. An *algebra* (X, x) for a (strong) endofunctor (F, st) consists of an object X together with a structure map $x : FX \to X$. Algebra *homomorphisms* $(X, x) \to (Y, y)$ are maps $h : X \to Y$ such that $h \circ x = y \circ Fh$. We write $F\text{-}alg$ for the corresponding category.

An *algebra* for a (strong) monad (T, st, η, μ) is a T-algebra with structure map satisfying identity and associativity laws with respect to η and μ. We write $T\text{-}Alg$ for the full subcategory of $T\text{-}alg$ so determined. (See *e.g.* [32, Section VI.2].)

Crucially, we will be interested in *left homomorphisms* [24] between algebras. These are given by maps $h : X \otimes P \to Y$ such that

$$\begin{array}{ccccc} F(X)\otimes P & \xrightarrow{st_{X,P}} & F(X\otimes P) & \xrightarrow{Fh} & FY \\ {\scriptstyle x\otimes P}\downarrow & & & & \downarrow{\scriptstyle y} \\ X\otimes P & & \xrightarrow[h]{} & & Y \end{array} \tag{1}$$

3 List objects

List objects have been mainly studied in the cartesian setting, see *e.g.* [21, 7, 33]. In the linear setting, the special case of Lawvere's NNO [29], which amounts to the list object on a unit object, has been considered in monoidal categories [38] (see also [6]) and in a syntactic calculus [2]. As for general list objects, as far as we know, they have only appeared indirectly under the guise of algebraically-free monoids (see Remark 3.2 below). The notion of list object is as expected and we make it explicit below.

▶ Remark. What we define under the name of *list object* is frequently referred to, in the cartesian setting, as a *parametrised list object*. We have adopted the shorter terminology for brevity and because we feel that it is the appropriate concept for the monoidal setting – especially in the extension to include algebraic structure put forward in Section 5.

▶ **Definition 3.1.** Let X be an object of a monoidal category $(I, \otimes)$.

1. A *list algebra* on X is an object L together with a pair of maps $\big(I \to L \leftarrow X \otimes L\big)$.
2. A *list object* on X is a list algebra $\big(I \xrightarrow{nil} \mathrm{L}(X) \xleftarrow{cons} X \otimes \mathrm{L}(X)\big)$ that is parametrised initial, in the sense that for every parametrised list algebra $(P \xrightarrow{n} L \xleftarrow{c} X \otimes L)$ there exists a unique mediating map $\mathrm{it}(n, c) : \mathrm{L}(X) \otimes P \to L$ such that

$$\begin{array}{ccccc} I\otimes P & \xrightarrow{nil\otimes P} & \mathrm{L}(X)\otimes P & \xleftarrow{cons\otimes P} & X\otimes \mathrm{L}(X)\otimes P \\ {\scriptstyle\cong}\downarrow & & {\scriptstyle \mathrm{it}(n,c)}\downarrow & & \downarrow{\scriptstyle X\otimes \mathrm{it}(n,c)} \\ P & \xrightarrow[n]{} & L & \xleftarrow[c]{} & X\otimes L \end{array}$$

Thus, list objects are universally characterised by their constructors and their support of definitions by pure iteration [43].

▶ Remark 3.2. In a monoidal category $(I, \otimes)$ with binary coproducts $(+)$ that are preserved by the endofunctor $(-)\otimes P$ for every object P, the notion of list object on an object X is equivalent to Kelly's notion of the algebraically-free monoid on the free pointed-object $(I \to I + X)$ on X, see [23, §23.1].

▶ **Example 3.3.** Our notion of list object on the unit object coincides with the notion of *LNNO (Left Natural Numbers Object)* of Paré and Román [38].

We summarise the algebraic structure of and provided by list objects.

▶ **Lemma 3.4.**

1. *Every list object on X with carrier $\mathrm{L}(X)$ provides a monoid structure with carrier $\mathrm{L}(X)$ that is the free monoid on X.*

2. *If the ambient category $\mathcal{C}$ has list objects, then the forgetful functor $Mon(\mathcal{C}) \to \mathcal{C}$ is monadic. This induces a list (or free monoid) monad* L.
3. *When the monoidal structure is cartesian the monad* L *is strong.*

▶ **Example 3.5.** The above explains the well-known construction of the free monad on an endofunctor by means of free algebras. Indeed, for an endofunctor F, a choice of free F-algebra $\big(X \xrightarrow{\eta_X} T_F X \xleftarrow{\varphi_X} F(T_F X)\big)$ on each object X determines a list object $(\mathrm{Id} \xrightarrow{\eta} T_F \xleftarrow{\varphi} F \circ T_F)$ on F in the category of endofunctors with the composition monoidal structure, inducing the free monad T_F on F. The case $F = \mathrm{Id}$ is precisely Burroni's notion of *NNO functor* that underlies his notion of Peano-Lawvere category [6] (consult also [38]).

▶ Remark. In general, the notions of list object and of free monoid do not agree. The former may not be available, as in the category of sets with the coproduct monoidal structure.

4 Parametrised initial algebras

As natural as the definition of list object is, it is also important to analyse its universal property through the theory of parametrised initial algebras. This section presents the relevant aspects of this theory.

▶ **Definition 4.1.** An *F-algebra* for a functor $F : \mathcal{D} \times \mathcal{C} \to \mathcal{C}$ is defined as a pair $\big((D, C), \varphi\big)$ with $\varphi : F(D, C) \to C$ in $\mathcal{C}$. F-algebras form a category *F-alg* in which a *homomorphism* $(D, C) \to (D', C')$ is a pair $(g : D \to D', f : C \to C')$ such that $f \circ \varphi = \varphi' \circ F(g, f)$.

▶ **Definition 4.2.** For a functor $F : \mathcal{D} \times \mathcal{C} \to \mathcal{C}$, the *initial-algebra functor* $\mu F : \mathcal{D} \to \mathcal{C}$ is defined, whenever possible, by choosing an initial $F(X, -)$-algebra $\big(\mu A.\, F(X, A), \varphi_X\big)$ and setting $\mu F(X) := \mu A.\, F(X, A)$. On morphisms, the action of μF on $f : X \to Y$ is uniquely determined by the fact that $\big(f, \mu F(f)\big)$ is an F-algebra homomorphism from $(\mu F(X), \varphi_X)$ to $(\mu F(Y), \varphi_Y)$.

▶ **Example 4.3.** Consider the functor $F : \mathcal{C} \times \mathcal{C} \to \mathcal{C} : (X, A) \mapsto I + X \otimes A$ in a monoidal category $(\mathcal{C}, I, \otimes)$ with binary coproducts $(+)$. In the presence of list objects $\mathrm{L}(X)$ on every $X \in \mathcal{C}$, the endofunctor μF is available and canonically isomorphic to the list-object endofunctor L.

We recall the basic iterative construction for building initial-algebra functors.

▶ **Lemma 4.4** ([1, 30])**.** *For every functor $F : \mathcal{D} \times \mathcal{C} \to \mathcal{C}$ where $\mathcal{C}$ has an initial object (0) and ω-colimits and $F_D := F(D, -)$ is ω-cocontinuous for every $D \in \mathcal{D}$, we have a functor $\mu F : \mathcal{D} \to \mathcal{C}$ defined by setting $\mu F(D)$ to be a chosen colimit of the ω-chain $\langle {F_D}^n(0 \to F_D 0)\rangle_{n \in \mathbb{N}}$. Moreover, if F is ω-cocontinuous then so is μF.*

We turn our attention to parametrised initiality. The general theory requires the framework of monoidal actions, see [10, Section I.1]. We do not dwell on this here; rather, we restrict attention to the special case relevant to the present development.

▶ **Definition 4.5.** For a monoidal category $(\mathcal{C}, I, \otimes)$, a strong functor $F : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, and an object $Z \in \mathcal{C}$, an initial $F(Z, -)$-algebra $(\mu A.\, F(Z, A), \varphi_Z)$ is said to be *parametrised initial* whenever for every object $P \in \mathcal{C}$ and algebra structure $\gamma : F(Z \otimes P, C) \to C$ on an object $C \in \mathcal{C}$ there is a unique map $u : \big(\mu A.\, F(Z, A)\big) \otimes P \to C$ making the following diagram

commute

$$\begin{array}{ccccc} F\big(Z, \mu A.\, F(Z,A)\big) \otimes P & \xrightarrow{st} & F\big(Z \otimes P, \big(\mu A.\, F(Z,A)\big) \otimes P\big) & \xrightarrow{F(Z\otimes P, u)} & F(Z \otimes P, C) \\ {\scriptstyle \varphi_Z \otimes P}\downarrow & & & & \downarrow{\scriptstyle \gamma} \\ \big(\mu A.\, F(Z,A)\big) \otimes P & & \xrightarrow[u]{} & & C \end{array}$$

▶ Remark. As is well-known, for a left-closed monoidal category (*i.e.*, one in which $(-) \otimes P$ has a right adjoint for all objects P), the notions of initiality and of parametrised initiality coincide.

▶ **Example 4.6.** Consider a monoidal category $(\mathcal{C}, I, \otimes)$ with binary coproducts $(+)$ that are preserved by every endofunctor $(-) \otimes P$ for $P \in \mathcal{C}$. For every object $X \in \mathcal{C}$, the functor $F_X : \mathcal{C} \times \mathcal{C} \to \mathcal{C} : (Z, A) \mapsto Z + X \otimes A$ is canonically strong and the notion of parametrised initial $F(I, -)$-algebra coincides with that of list object on X.

We now provide a conceptual framework for parametrised iteratively-constructed initial algebras. Our main technical tool is the following *lax-uniformity* property of initial algebra functors. (For uniformity in a related 2-dimensional setting see [9, Section 7.3.2].)

▶ **Theorem 4.7.** *Consider a diagram of categories, functors, and a natural transformation as on the left below*

$$\begin{array}{ccc} \mathcal{D} \times \mathcal{C} & \xrightarrow{F} & \mathcal{C} \\ {\scriptstyle K\times J}\downarrow & \overset{h}{\Leftarrow} & \downarrow{\scriptstyle J} \\ \mathcal{B} \times \mathcal{A} & \xrightarrow[G]{} & \mathcal{A} \end{array} \qquad \begin{array}{ccc} \mathcal{D} & \xrightarrow{\mu F} & \mathcal{C} \\ {\scriptstyle K}\downarrow & \overset{\mu h}{\Leftarrow} & \downarrow{\scriptstyle J} \\ \mathcal{B} & \xrightarrow[\mu G]{} & \mathcal{A} \end{array} \tag{2}$$

Assume that the categories $\mathcal{C}, \mathcal{A}$ both have an initial object and ω-colimits, and that the functors $F(D, -), G(B, -), J$ are ω-cocontinuous for all $D \in \mathcal{D}$ and $B \in \mathcal{B}$. If J also preserves initial objects, then for all algebras $\alpha : G(KD, A) \to A$ there exists a unique map $\alpha^\star : J(\mu F(D)) \to A$ such that

$$\begin{array}{ccccc} JF(D, \mu F(D)) & \xrightarrow{h_{D,\mu F(D)}} & G\big(KD, J(\mu F(D))\big) & \xrightarrow{G(KD, \alpha^\star)} & G(KD, A) \\ {\scriptstyle J\cong}\downarrow & & & & \downarrow{\scriptstyle \alpha} \\ J(\mu F(D)) & & \xrightarrow[\alpha^\star]{} & & A \end{array}$$

and we have the situation as on the right in (2) above with

$$\mu h_D := \big(G(KD, \mu G(KD)) \xrightarrow{\cong} \mu G(KD)\big)^\star .$$

The lemma below follows by applying the theorem above to the diagram

$$\begin{array}{ccc} \mathcal{C} \times \mathcal{C} & \xrightarrow{F} & \mathcal{C} \\ {\scriptstyle \mathrm{Id}\times(-\otimes P)}\downarrow & \overset{st}{\Leftarrow} & \downarrow{\scriptstyle -\otimes P} \\ \mathcal{C} \times \mathcal{C} & \xrightarrow[F(-\otimes P, =)]{} & \mathcal{C} \end{array}$$

for each $P \in \mathcal{C}$.

▶ **Lemma 4.8.** *Let $(\mathcal{C}, I, \otimes)$ be a monoidal category with an initial object and ω-colimits, both preserved by the endofunctor $(-) \otimes P$ for each $P \in \mathcal{C}$. For an ω-cocontinuous strong endofunctor $(F, st) : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, every initial $F(Z, -)$-algebra for $Z \in \mathcal{C}$ is parametrised initial.*

From this lemma and Example 4.6, we attain conditions for (not necessarily left-closed) monoidal categories under which list objects arise as iteratively-constructed initial algebras.

▶ **Corollary 4.9.** *Let $(\mathcal{C}, I, \otimes)$ be a monoidal category with finite coproducts $(0, +)$ and ω-colimits, both preserved by the endofunctor $(-) \otimes P$ for all $P \in \mathcal{C}$. For $X \in \mathcal{C}$, if the endofunctor $X \otimes (-)$ is ω-cocontinuous, then the initial $(I + X \otimes -)$-algebra gives a list-object structure on X.*

5 List objects and monoids with algebraic structure

We have observed that list objects provide free monoids (Lemma 3.4). In this section, we generalise this fact to encompass notions of list object and monoid that are equipped with algebraic structure.

List objects with algebraic structure. We equip list objects with a monadic algebra structure. Crucially, we use this algebraic structure to refine their defining universal property. This is the key concept of the paper.

▶ **Definition 5.1.** Let (T, st) be a strong monad on a monoidal category $(I, \otimes)$.

1. A *T-list algebra* on an object X consists of an object A together with a list algebra $\big(I \to A \leftarrow X \otimes A\big)$ and a T-algebra structure $TA \to A$.
2. A *T-list object* on an object X is a parametrised initial T-list algebra. Explicitly, it is a T-list algebra as on the left below

$$\begin{array}{ccccc} & & T(\mathrm{M}(X)) & & \\ & & \downarrow{\scriptstyle \tau} & & \\ I & \xrightarrow[nil]{} & \mathrm{M}(X) & \xleftarrow[cons]{} & X \otimes \mathrm{M}(X) \end{array} \qquad \begin{array}{ccccc} & & TA & & \\ & & \downarrow{\scriptstyle \alpha} & & \\ P & \xrightarrow[n]{} & A & \xleftarrow[c]{} & X \otimes A \end{array}$$

such that for every parametrised T-list algebra as on the right above there exists a unique mediating map $\mathrm{it}(n, c, \alpha) : \mathrm{M}(X) \otimes P \to A$ such that

$$\begin{array}{ccccc} I \otimes P & \xrightarrow{nil \otimes P} & \mathrm{M}(X) \otimes P & \xleftarrow{cons \otimes P} & X \otimes \mathrm{M}(X) \otimes P \\ {\scriptstyle \cong}\downarrow & & \downarrow{\scriptstyle \mathrm{it}(n,c,\alpha)} & & \downarrow{\scriptstyle X \otimes \mathrm{it}(n,c,\alpha)} \\ P & \xrightarrow[p]{} & A & \xleftarrow[c]{} & X \otimes A \end{array}$$

$$\begin{array}{ccccc} T\big(\mathrm{M}(X)\big) \otimes P & \xrightarrow{st_{\mathrm{M}(X),P}} & T(\mathrm{M}(X) \otimes P) & \xrightarrow{T(\mathrm{it}(n,c,\alpha))} & TA \\ {\scriptstyle \tau \otimes P}\downarrow & & & & \downarrow{\scriptstyle \alpha} \\ \mathrm{M}(X) \otimes P & & \xrightarrow[\mathrm{it}(n,c,\alpha)]{} & & A \end{array}$$

Thus, definitions by pure iteration in this setting are required to be left algebra homomorphisms (recall (1)).

Monoids with algebraic structure. Generalising from [15], we define the notion of monoid equipped with compatible monadic algebraic structure. This has also recently been considered by Piróg [39] under the terminology of Eilenberg-Moore monoid.

▶ **Definition 5.2.** Let (T, st) be a strong monad on a monoidal category $(\mathcal{C}, I, \otimes)$. A *$T$-monoid* is an object C equipped with a monoid structure $(I \xrightarrow{e} C \xleftarrow{m} C \otimes C)$ and a T-algebra structure $(c : TC \to C)$ compatible in the sense that the monoid multiplication is a left homomorphism; that is, the following diagram commutes:

$$\begin{array}{ccccc} T(C) \otimes C & \xrightarrow{st_{C,C}} & T(C \otimes C) & \xrightarrow{Tm} & TC \\ {\scriptstyle c\otimes C}\downarrow & & & & \downarrow{\scriptstyle c} \\ C \otimes C & & \xrightarrow[m]{} & & C \end{array}$$

Homomorphisms of T-monoids are morphisms of the underlying objects that are both monoid and T-algebra homomorphisms. We write $T\text{-}Mon(\mathcal{C})$ for the category of T-monoids and their homomorphisms.

▶ Remark. The construction $T\text{-}Mon(\mathcal{C})$ generalises both the category of algebras for a monad $T\text{-}Alg$ and the category of monoids $Mon(\mathcal{C})$. The former is available for any category $\mathcal{C}$ with finite coproducts by taking these as the monoidal structure; the latter arises by restricting attention to identity monads.

The notion of T-monoid is essentially an extension of the notion of F-monoid for a strong endofunctor F on a monoidal category $\mathcal{C}$, introduced in [15, Section 4], to incorporate further algebraic structure. The original notion is typically subsumed by the notion of T_F-monoid for T_F the monad on $\mathcal{C}$ induced by a left adjoint to the forgetful functor $F\text{-}alg \to \mathcal{C}$. Indeed, consider the following proposition.

▶ **Proposition 5.3.** *Let $(\mathcal{C}, I, \otimes)$ be a monoidal category and F a strong endofunctor on it.*

1. *If the forgetful functor $F\text{-}alg \to \mathcal{C}$ has a left adjoint then it is monadic.*
2. *If $\otimes$ is left-closed and the forgetful functor $F\text{-}alg \to \mathcal{C}$ has a left adjoint then the induced monad T_F is canonically strong.*
3. *Assuming that $\mathcal{C}$ has finite coproducts and ω-colimits that are preserved by $(-) \otimes P$ for all $P \in \mathcal{C}$, if F is ω-cocontinuous then the forgetful functor $F\text{-}alg \to \mathcal{C}$ has a left adjoint and the induced monad T_F is canonically strong.*
4. *In either of the situations (2) and (3) above, the notions of T_F-monoid and of F-monoid coincide.*

Wolff [48] noticed that, for T a strong monad on a monoidal category $(I, \otimes)$, TI is a monoid. The notion of T-monoid allows us to sharpen this result.

▶ **Lemma 5.4.** *For every strong monad (T, st, η, μ) on a monoidal category $(\mathcal{C}, I, \otimes)$, the free T-algebra (TI, μ_I) equipped with unit $\eta_I : I \to TI$ and multiplication*

$$(TI) \otimes (TI) \xrightarrow{st_{I,TI}} T(I \otimes TI) \cong T(TI) \xrightarrow{\mu_I} TI$$

is a T-monoid. In fact, it is initial in $T\text{-}Mon(\mathcal{C})$.

The original example in which the lemma above was applied essentially considered the monad freely generated by a binding-signature endofunctor. In that context, the initiality property yields initial-algebra semantics for abstract syntax with variable binding and substitution. See [15, 10] for details.

For further examples of T-monoids consider the following.

▶ **Proposition 5.5.** *For every monoid M in a monoidal category $(\mathcal{C}, I, \otimes)$, we have a strong monad $M^{\otimes}$ $\big($with underlying endofunctor $M \otimes (-)\big)$ for which $M^{\otimes}$-Alg is the category of left M-actions, and $M^{\otimes}$-Mon$(\mathcal{C})$ is isomorphic to the slice category $M/Mon(\mathcal{C})$.*

***T*-list objects induce free *T*-monoids.** For a strong ω-cocontinuous monad T on a finitely and ω-chain cocomplete monoidal category $(I, \otimes)$ in which $\otimes$ is ω-cocontinuous, free T-monoids can be shown to exist (for instance, by applying the theory of [13]). However, no explicit description for this construction is available. The main purpose of this section is to establish one under practical hypotheses.

First we observe that the situation for list objects generalises in the expected way (compare Lemma 3.4).

▶ **Lemma 5.6.** *Let T be a strong monad on a monoidal category $\mathcal{C}$.*

1. *T-list objects induce free T-monoids.*
2. *If $\mathcal{C}$ has T-list objects then the forgetful functor T-Mon$(\mathcal{C}) \to \mathcal{C}$ is monadic inducing a free T-monoid (or T-list) monad M_T.*
3. *When the monoidal structrure is cartesian closed, the monad M_T is strong.*

The proof that list objects induce free monoids relies on parametrised initiality to define the multiplication map by iteration and on the uniqueness property of mediating maps to show that the monoid laws and universal property hold. The proof of Lemma 5.6(1) follows the same approach, except one needs to check the relevant maps are left homomorphisms.

In the light of the lemma, the problem of constructing free T-monoids can be reduced to that of constructing T-list objects. For an object X in a monoidal category $(\mathcal{C}, I, \otimes)$ with binary coproducts $(+)$, we introduce the functor

$$F(P, C) := T(P + X \otimes C) \ ,$$

consider a parameterised initial $F(I, -)$-algebra

$$\mathrm{M}X := \mu A.\, T(I + X \otimes A) \ ,$$

and proceed to equip it with a T-list object structure on X. We construct the list algebra $(I \xrightarrow{nil} \mathrm{M}X \xleftarrow{cons} X \otimes \mathrm{M}X)$ as

$$nil := \big(I \xrightarrow{\mathrm{inl}} I + X \otimes \mathrm{M}X \xrightarrow{\eta} T(I + X \otimes \mathrm{M}X) \xrightarrow{\varphi} \mathrm{M}X\big)$$

and

$$cons := \big(X \otimes \mathrm{M}X \xrightarrow{\mathrm{inr}} I + X \otimes \mathrm{M}X \xrightarrow{\eta} T(I + X \otimes \mathrm{M}X) \xrightarrow{\varphi} \mathrm{M}X\big)$$

where $\varphi : T(I + X \otimes \mathrm{M}X) \xrightarrow{\cong} \mathrm{M}X$ is the structure map for $\mathrm{M}X$. This has a T-algebra structure, given by

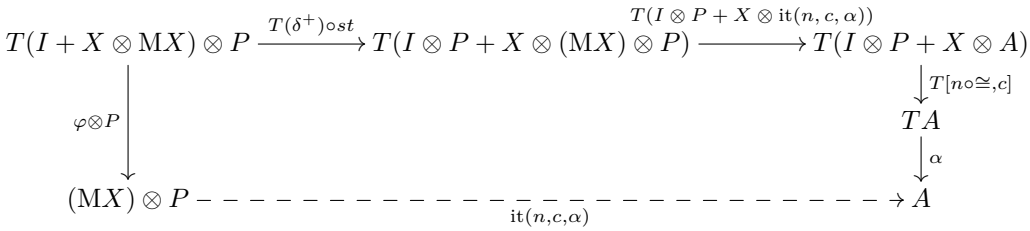
$$\widetilde{\mu} := \big(\ T(\mathrm{M}X) \xrightarrow[\cong]{T\varphi^{-1}} TT(I + X \otimes \mathrm{M}X) \xrightarrow{\mu} T(I + X \otimes \mathrm{M}X) \xrightarrow[\cong]{\varphi} \mathrm{M}X\ \big) \ .$$

▶ **Lemma 5.7.** *For a strong monad (T, st) on a monoidal category $(I, \otimes)$ with binary coproducts $(+)$ preserved by the endofunctor $(-) \otimes P$ for every object P, the structure*

$$\begin{array}{ccccc} & & T(\mathrm{M}X) & & \\ & & \big\downarrow{\scriptstyle \widetilde{\mu}} & & \\ I & \xrightarrow[nil]{} & \mathrm{M}X & \xleftarrow[cons]{} & X \otimes \mathrm{M}X \end{array} \tag{3}$$

is a T-list object.

In particular, the mediating map $\mathrm{it}(n, c, \alpha) : (\mathrm{M}X) \otimes P \to A$ to a parametrised T-list algebra $(P \xrightarrow{n} A \xleftarrow{c} X \otimes A, TA \xrightarrow{\alpha} A)$ arises by parametrised initiality (recall Definition 4.5) as follows



where δ^+ denotes the inverse of the canonical map $(- \otimes P) + (= \otimes P) \to (- + =) \otimes P$.

We thus have the following result.

▶ **Theorem 5.8.** *Let T be a strong monad on a monoidal category $(\mathcal{C}, I, \otimes)$ with binary coproducts $(+)$. If*

1. *for every $P \in \mathcal{C}$, the endofunctor $(-) \otimes P$ preserves binary coproducts, and*
2. *every functor $F_X(-, =) := T(- + X \otimes =)$ for $X \in \mathcal{C}$ has a parametrised initial $F_X(I, -)$-algebra*

then $\mathcal{C}$ has T-list objects as in (3) above and, thereby, the free T-monoid monad M_T.

▶ Remark. We have that $\mu A.\, T(I + X \otimes A) \cong T\big(\mu A.\, (I + X \otimes TA)\big)$ by the Rolling Rule [3, Theorem 4], which gives the description of free T-monoids of Piróg [39].

▶ **Example 5.9.**

1. The free Id-monoid monad is the list monad.
2. One may apply Theorem 5.8 in the situation of Proposition 5.3 to describe the free F-monoid for a strong endofunctor F. This is given by

   $$\mu A.\, T_F(I + X \otimes A) = \mu A.\, \mu B.\, I + X \otimes A + F(B)$$

   or, by the Diagonal Rule [3, Theorem 16], equivalently by

   $$\mathrm{M}_{T_F}(X) := \mu C.\, I + X \otimes C + F(C)\,.$$

   Thus, we recover a slightly restricted version of a corresponding result for endofunctors with *pointed strength* of Fiore [10, Theorems 2 and 3] (see also Section 7.3 below). In the context of the motivating application of [10], this yields initial-algebra semantics for abstract syntax with variable binding and parameterised metavariables, with their associated operations of capture-avoiding substitution and of meta-substitution. See [17, 10] for details.

For our applications, we are interested in using Theorem 5.8 in two distinct concrete scenarios, encapsulated in the corollaries below.

▶ **Corollary 5.10.** *Let T be a strong monad on a left-closed monoidal category $(\mathcal{C}, I, \otimes)$ with binary coproducts $(+)$. If every endofunctor $T(I + X \otimes -)$ for $X \in \mathcal{C}$ has an initial algebra then $\mathcal{C}$ has T-list objects and, thereby, the free T-monoid monad M_T.*

▶ **Corollary 5.11.** *Let T be a strong monad on a monoidal category $(\mathcal{C}, I, \otimes)$ with finite coproducts $(0, +)$ and ω-colimits. If*

1. *for every $P \in \mathcal{C}$, the endofunctor $(-) \otimes P$ preserves finite coproducts, and*
2. *the functors $\otimes$ and T are ω-cocontinuous*

then $\mathcal{C}$ has T-list objects and, thereby, the free T-monoid monad M_T.

▶ **Example 5.12.** In view of Proposition 5.5 and Corollary 5.10, for every monoid M in a left-closed monoidal category $(\mathcal{C}, I, \otimes)$ with binary coproducts $(+)$ the free M-pointed object in $Mon(\mathcal{C})$ on an object $X \in \mathcal{C}$ is given by $\mu A.\, M \otimes (I + X \otimes A)$.

6 Strong algebraic structure

Our discussion so far has considered algebraic structure abstractly as encapsulated by the notion of monad. Our main interest is in monads that are strong, and this section examines these as they arise from algebraic theories. Our aim here is not to treat these in full generality (as *e.g.* in the enriched setting [41]) but to focus attention on two traditionally important contexts: cartesian and monoidal. By way of introduction, Section 6.1 recalls the setting of (cartesian) algebraic theories as studied in universal algebra [28, 8, 40]. Then, in Section 6.2, we develop the operadic [5, 36] or linear [27] setting. Our analysis unveils the notion of nsr (or near semiring) category (Definition 6.2), a categorification of the algebraic structure of near semiring [47], as the basic categorical framework in which to consider monoidal (or linear) algebraic models.

6.1 Algebraic theories

A *Lawvere theory* [28] is seen in universal algebra as an *abstract clone* of operations (see *e.g.* [8, 40]). This is a family of sets $O = \{O_n\}_{n\in\mathbb{N}}$ together with *variables* $x_i^{(n)} \in O_n$ $(1 \leq i \leq n \in \mathbb{N})$ and *substitution* operations $s_{m,n} : O_m \times {O_n}^m \to O_n$ $(m, n \in \mathbb{N})$ subject to identity and associativity laws. A *map* $f : O \to O'$ of abstract clones is a family of functions $f_n : O_n \to O'_n$ $(n \in \mathbb{N})$ preserving variables and commuting with substitution.

A main example of an abstract clone is the *concrete clone* of operations (or clone of endomorphisms) on an object in a cartesian category $\mathcal{C}$. For an object $X \in \mathcal{C}$ this is defined as $E(X) = \{\, \mathcal{C}(X^n, X) \,\}_{n\in\mathbb{N}}$ with variables given by projections and substitution by composition. An *O-algebra* (or model) of an abstract clone O is an object X together with a structure map $O \to E(X)$ of abstract clones. An algebra *homomorphism* $h : (X, x) \to (Y, y)$ is a map $h : X \to Y$ such that $h \circ x_o = y_o \circ h^n : X^n \to Y$ for all $n \in \mathbb{N}$ and $o \in O_n$. We write *O-Alg* for the corresponding category. If $\mathcal{C}$ is cocomplete and the product $(\times)$ is separately cocontinuous in each argument, the forgetful functor *O-Alg* $\to \mathcal{C}$ is monadic and the induced monad T_O is given on an object $X \in \mathcal{C}$ by the coequaliser

$$\coprod_{m,n\in\mathbb{N}} \coprod_{\rho\in\mathbb{F}([m],[n])} O_m \cdot X^n \underset{[\iota_n \circ (O_\rho \cdot X^n)]_{\rho:[m]\to[n]}}{\overset{[\iota_m \circ (O_m \cdot X^\rho)]_{\rho:[m]\to[n]}}{\rightrightarrows}} \coprod_{\ell\in\mathbb{F}} O_\ell \cdot X^\ell \twoheadrightarrow T_O(X)$$

where we write $\cdot$ for the functor $\mathbf{Set} \times \mathcal{C} \to \mathcal{C} : (I, C) \mapsto \coprod_{i\in I} C$, and where we let $\mathbb{F}$ denote the category of finite cardinals $[n] := \{1, \ldots, n\}$ $(n \in \mathbb{N})$ and functions between them and set $O_\rho := s_{m,n}\big(-, (x_{\rho 1}^{(n)}, \ldots, x_{\rho m}^{(n)})\big) : O_m \to O_n$ for all $\rho \in \mathbb{F}([m], [n])$. Intuitively, thus, the coequaliser identifies $\iota_o(-_{\rho 1}, \ldots, -_{\rho m})$ and $\iota_{O_\rho(o)}(-_1, \ldots, -_n)$ for all $m, n \in \mathbb{N}$, $\rho \in \mathbb{F}([m], [n])$, and $o \in O_m$.

Important to our concerns here, when $\mathcal{C}$ is furthermore monoidal and the tensor product $(\otimes)$ is left cocontinuous, the free monad T_O is canonically strong, with strength components given by the unique map making the following diagram commute

$$\begin{array}{ccc} \coprod_{\ell\in\mathbb{F}} O_\ell \cdot (X^\ell \otimes P) & \twoheadrightarrow & T_O(X) \otimes P \\ \downarrow{\scriptstyle \coprod_{\ell\in\mathbb{F}} O_\ell \cdot \langle \pi_1 \otimes P, \ldots, \pi_\ell \otimes P \rangle} & & \downarrow \\ \coprod_{\ell\in\mathbb{F}} O_\ell \cdot (X \otimes P)^\ell & \twoheadrightarrow & T_O(X \otimes P) \end{array}$$

We can therefore consider T_O-monoids. At any rate, one can also give an equivalent direct definition for abstract clones (or for their equational presentations).

▶ **Definition 6.1.** An O-monoid, for an abstract clone O in a monoidal category with finite products, is an object C equipped with a monoid structure $(I \xrightarrow{e} C \xleftarrow{m} C \otimes C)$ and an O-algebra structure $\gamma : O \to E(C)$ compatible in the sense that the monoid multiplication is a left algebra homomorphism; that is, the diagram

$$\begin{array}{ccccc} C^n \otimes C & \xrightarrow{\langle \pi_1 \otimes C, \ldots, \pi_n \otimes C \rangle} & (C \otimes C)^n & \xrightarrow{m^n} & C^n \\ {\scriptstyle \gamma_o \otimes C}\downarrow & & & & \downarrow{\scriptstyle \gamma_o} \\ C \otimes C & & \xrightarrow[m]{} & & C \end{array}$$

commutes for all $n \in \mathbb{N}$ and $o \in O_n$.

In the next section we shall see that this construction falls within a more general framework.

6.2 Linear algebraic theories

Operads are a well-established formalism for investigating monoidal or linear algebraic structure, see *e.g.* [34]. In their basic form, they come in two guises: plain and symmetric. Here, for simplicity, we limit the discussion to the plain case; the development can be suitably adapted to the symmetric case. A *plain operad* is a family of sets $O = \{ O_n \}_{n \in \mathbb{N}}$ together with a *variable* $x \in O_1$ and *linear substitution* operations $s_\ell^{(n_1,\ldots,n_\ell)} : O_\ell \times \prod_{i \in [\ell]} O_{n_i} \to O_{\Sigma_{i \in [\ell]} n_i}$ $(\ell, n_1, \ldots, n_\ell \in \mathbb{N})$ subject to identity and associative laws. A map $f : O \to O'$ of plain operads is a family of functions $f_n : O_n \to O'_n$ $(n \in \mathbb{N})$ preserving the variable and commuting with substitution. (See *e.g.* [31, Chapter 2]).

A main example of a plain operad is the *endomorphism operad* on an object in a monoidal category. For an object X in a monoidal category $(\mathcal{C}, J, *)$ this is defined as $E(X) = \{ \mathcal{C}(X^{*n}, X) \}_{n \in \mathbb{N}}$ with variable given by the identity and substitution by multi-composition. The category of algebras for an operad O is defined analogously to that for abstract clones, and equally denoted O-*Alg*. Without going into details, the theory of [13] may be applied to show that when $\mathcal{C}$ is cocomplete and $*$ is ω-cocontinuous, the forgetful functor O-$Alg \to \mathcal{C}$ is monadic. We are interested in establishing a framework in which the induced free O-algebra monads are strong. This is provided by the following notion.

▶ **Definition 6.2.** An *nsr-category* is a category equipped with two monoidal structures, $(I, \otimes)$ and $(J, *, \lambda, \rho, \alpha)$, together with a *distributive structure* of the former over the latter specified by $\otimes$-strengths for J and $*$

$$\delta_P^{(0)} : J \otimes P \to J \ , \quad \delta_{A,B,P}^{(2)} : (A * B) \otimes P \to (A \otimes P) * (B \otimes P)$$

such that λ, ρ, α are $\otimes$-strong.

▶ Remark. As in [42], the above terminology is motivated by the algebraic structure known as a *near semiring* [47], consisting of two monoid structures on the same carrier that are subject to one-sided annihilation and distributivity laws of one structure over the other.

▶ **Example 6.3. 1.** For a monoidal category $(\mathcal{C}, I, \otimes)$ with finite coproducts $(0, +)$, the opposite category $\mathcal{C}^{\mathrm{op}}$ has a canonical distributive structure

$$0 \otimes P \leftarrow 0 \ , \quad (A + B) \otimes P \leftarrow (A \otimes P) + (B \otimes P)$$

of $\otimes$ over $+$. Hence, whenever the above maps are invertible, we also have a distributive structure of $\otimes$ over $+$ in $\mathcal{C}$.

2. Cartesian monoidal categories have a distributive structure of the monoidal structure over the cartesian structure. This was implicitly used in the previous section.
3. The category of endofunctors of a monoidal category has a distributive structure of the composition monoidal structure over the pointwise monoidal structure.
4. Joyal's category of combinatorial species of structures [20] forms an nsr-category with the substitution and multiplication monoidal structures. This was implicitly used in [11].

The natural ambient universe for monoids with linear algebraic structure (either given by an operad or their equational presentations) is that of nsr-categories.

▶ **Definition 6.4.** An O-monoid, for an operad O in an nsr-category $(I, \otimes, J, *, \delta)$, is an object C equipped with a $\otimes$-monoid structure $(I \xrightarrow{e} C \xleftarrow{m} C \otimes C)$ and an O-algebra $*$-structure $\gamma : O \to E(C)$, compatible in the sense that the monoid multiplication is a left algebra homomorphism; that is, the diagram

$$\begin{array}{ccccc} C^{*n} \otimes C & \xrightarrow{\delta^{(n)}} & (C \otimes C)^{*n} & \xrightarrow{m^{*n}} & C^{*n} \\ {\scriptstyle \gamma_o \otimes C}\downarrow & & & & \downarrow{\scriptstyle \gamma_o} \\ C \otimes C & & \xrightarrow[m]{} & & C \end{array}$$

commutes for all $n \in \mathbb{N}$ and $o \in O_n$, where $\delta^{(n)}_{C_1,\ldots,C_n,P}$ stands for the iterated strength $(C_1 * \cdots * C_n) \otimes P \to (C_1 \otimes P) * \cdots * (C_n \otimes P)$.

▶ Remark 6.5. In well-behaved settings, one may typically rephrase O-monoids in terms of T-monoids. Specifically, in a cocomplete nsr-category $(\mathcal{C}, \otimes, *)$ with $\otimes$ left-cocontinuous and $*$ ω-cocontinuous, the forgetful functor $O\text{-}Alg \to \mathcal{C}$ is monadic and the induced free O-algebra monad T_O is $\otimes$-strong. Furthermore, the notions of T_O-monoid and of O-monoid coincide. For instance, this is the case in the example featured in [11, Example 5.13(3)].

We conclude the section by discussing an important concrete example.

▶ **Lemma 6.6.** *Let $(\mathcal{C}, \otimes, *)$ be an nsr-category.*

1. *Assuming that $\mathcal{C}$ has $*$-list objects and that $\otimes$ is left-closed, the $*$-list monad L_* (arising from Lemma 3.4) is $\otimes$-strong.*
2. *Assuming that $\mathcal{C}$ has finite coproducts and ω-colimits, that $\otimes$ is left ω-cocontinuous, that $*$ is ω-cocontinuous, and that $(-) * P$ preserves finite coproducts for all $P \in \mathcal{C}$, the $*$-list monad L_* (arising from Corollary 4.9 and Lemma 3.4) is $\otimes$-strong.*

▶ **Example 6.7.** The notion of O-monoid for O the theory of monoids in an nsr-category $(\mathcal{C}, I, \otimes, J, *, \delta)$ amounts to an object C equipped with two monoid structures $(I \xrightarrow{1} C \xleftarrow{\times} C \otimes C)$ and $(J \xrightarrow{0} C \xleftarrow{+} C * C)$ such that

$$\begin{array}{ccc} J \otimes C & \xrightarrow{\delta^{(0)}_C} & J \\ {\scriptstyle 0 \otimes C}\downarrow & & \downarrow{\scriptstyle 0} \\ C \otimes C & \xrightarrow[\times]{} & C \end{array} \qquad \begin{array}{ccccc} (C * C) \otimes C & \xrightarrow{\delta^{(2)}_{C,C,C}} & (C \otimes C) * (C \otimes C) & \xrightarrow{\times * \times} & C * C \\ {\scriptstyle + \otimes C}\downarrow & & & & \downarrow{\scriptstyle +} \\ C \otimes C & & \xrightarrow[\times]{} & & C \end{array}$$

These two diagrams respectively express one-sided annihilation and distributivity laws. As such, we refer to these structures as *nsr (or near-semiring) objects*. Nsr-objects in nsr-categories have already been considered by Rivas *et al.* [42].

For a $\otimes$-strong $*$-list monad L_* (arising, for instance, as in Lemma 6.6), L_*-monoids are precisely nsr-objects. In the context of Theorem 5.8 we therefore have that the free nsr-object on an object X is given by

$$\mathrm{Nsr}(X) := \mu A.\, \mathrm{L}_*(I + X \otimes A)\,.$$

▶ **Example 6.8.** For a strong monad T on a monoidal category $(\mathcal{C}, I, \otimes)$, we have a strong monad T° (with underlying endofunctor $T \circ (-)$) on the monoidal category $(\mathrm{Endo}(\mathcal{C}), I, \otimes)$ of endofunctors with the pointwise monoidal structure, and $\mathrm{M}_{T^\circ}(\mathrm{Id}) = \mathrm{M}_T$; that is, the free T°-monoid on the identity is the free T-monoid monad. Furthermore, it is an nsr-object in the nsr-category $(\mathrm{Endo}(\mathcal{C}), \mathrm{Id}, \circ, I, \otimes)$ (recall Example 6.3(3)).

7 Applications and related work

Structures akin to the ones studied in this paper have been considered in a variety of contexts. We have already mentioned the algebraic approach to abstract syntax and variable binding of [15, 10] (see also [18]); we also touch upon this below. Before that, we will discuss applications to programming theory, in semantics and functional programming, and to higher-dimensional algebra, in the opetopic approach [4, 45]. Our work connects these developments and opens up possibilities for interaction between them.

7.1 Semantics and programming

Let us uncover some examples of T-list objects in semantics and programming.

Natural Numbers Objects. Following Lawvere [29], we refer to the T-list object on the unit object as a T-NNO. This general notion provides a uniform sense in which the various domains of natural numbers that are needed in denotational semantics are Natural Numbers Objects for suitable algebraic notions of primitive recursion. Indeed, Corollary 5.11 has the following consequences in the category of cpos and continuous functions: the NNO for the cartesian monoidal structure is the *natural numbers* $\mu A.\, 1 + A$ and the lifting-NNO for the cartesian monoidal structure is the *lazy natural numbers* $\mu A.\, (1 + A)_\perp$. Moreover, the lifting-NNO for the cocartesian monoidal structure is the *strict natural numbers* $\mu A.\, A_\perp$.

List transformer. Corollary 5.10 gives a universal property for the monadic *list transformer* of Jaskelioff [19, Example 4.7] (see also [16]): it is a T-list object construction and the free T-monoid monad in the bicartesian closed setting. Indeed, assuming the needed initial algebras exist, the list transformer $\mathrm{Lt}(T)$ of a strong monad T on a cartesian closed category with binary coproducts is defined as $\mathrm{Lt}(T)\, X := \mu A.\, T(1 + X \times A)$. This result was first established by Kammar [22] using a type-theoretic internal language. It follows from Example 6.8 that the list transformer produces instances of the Haskell MonadPlus type class also discussed below.

7.2 Free monoids with linear algebraic structure in nsr-categories

The examples of this section arise from the linear algebraic framework of Section 6.2, where instances of free T-monoids have been considered in functional programming and higher-dimensional algebra.

7.2.1 Functional programming

We consider an nsr-category $(\mathcal{C}, I, \otimes, J, *)$ with left-closed tensor products and binary coproducts, placing ourselves in the setting of Corollary 5.10. This is the setting assumed in the context of functional programming, say in Haskell.

For nsr-categories with structure as in Example 6.3(2) (*i.e.* with $(J, *) = (1, \times)$), Rivas *et al.* [42] constructed free nsr-objects as in Example 6.7. Specialising this construction to the nsr-category of endofunctors with structure $(\mathrm{Id}, \circ, 1, \times)$ (recall Example 6.3(3)) over a cartesian category $(1, \times)$ they obtained, and programmed with, instances of the Haskell MonadPlus type class freely generated from an endofunctor by means of the recursive type $\mathrm{Mp}(F)\, X := \mu A.\, \mathrm{List}(X + FA)$. Further examples in this direction may be found in [46].

For the purpose of combinatorial search, Spivey [44] proposed extending the Haskell MonadPlus type class to a Bunch type class by adding a unary *wrap* operation subject to an equation. In our linear algebraic framework of Section 6.2, this precisely amounts to extending the theory of monoids with an extra unary operation. The free algebras for this theory are given by $T(X) := \mu A.\, J + (X + A) * A$ with induced free T-monoids $\mathrm{Bun}(X) := \mu A.\, J + (I + X \otimes A + A) * A$. Specialising these constructions to the monadic setting of functional programming, one obtains freely generated instances of the Haskell Bunch type class by means of the recursive type $\mathrm{Bun}(F)\, X := \mu A.\, 1 + X \times A + F(A) \times A + A \times A$.

7.2.2 Higher-dimensional algebra

Let $(\mathcal{C}, I, \otimes, J, *)$ be a cocomplete nsr-category (Definition 6.2) where $\otimes$ is left cocontinuous and right ω-cocontinuous and $*$ is ω-cocontinuous. Then, the free O-algebra monad T_O (Remark 6.5) is ω-cocontinuous and Corollary 5.11 applies.

Restricting attention to the theory of monoids, it is enough to require that both tensor products be ω-cocontinuous and that they preserve finite coproducts in their first argument for free monoids to be iteratively constructed as list objects. In fact, this is precisely the context in which the work of Szawiel and Zawadowski [45] takes place. In particular, they define the *web monoid* as $\mathrm{L}_*(I)$ and establish a universal property for it. Our results sharpen and generalise theirs: by establishing that the web monoid is the initial nsr-object $\mathrm{M}_{\mathrm{L}_*}(0)$, and by providing a general construction of free nsr-objects $\mathrm{M}_{\mathrm{L}_*}(X)$.

7.3 Second-order algebraic theories

The application of the theory of monoids with algebraic structure for an endofunctor or a monad in the study of abstract syntax with variable binding [15] and with parametrised metavariables [17, 10] crucially requires a slightly more general theory than the one expounded upon here (recall Example 5.9(2)). Indeed, because of the presence of binding and the need for capture-avoidance in substitution, the semantic endofunctors and monads freely generated from the syntax only admit a *pointed strength* (see [10] and [14]); that is, a strength of the form

$$st_{X,I\to P} : T(X) \otimes P \to T(X \otimes P)$$

only available for parameters P equipped with a point $(I \to P)$. The results of the present work generalise not only to this setting but, in fact, to a more comprehensive one involving monoidal actions (see [10]). As such, they will be presented elsewhere.

8 Concluding remarks

We have introduced the notions of T-list object and T-monoid, developing some of their related basic theory and discussing applications. In particular, in Section 5, we have established that T-list objects yield free T-monoids and have provided an explicit description for them:

$$\mathrm{M}_T(X) := \mu A.\, T(I + X \otimes A)$$

that we have shown to be available in common practical scenarios. Our theory captures all the examples of this construction that we are aware of. In this respect, the study of parametrised initiality of Section 4 is crucial for cases where the tensor product is not left-closed. Considering compatible algebraic and monoid structures from the perspective of T-monoids leads naturally to the notion of nsr-category. These categories provide a common framework for applications in a diverse range of settings, some of which we have presented in Sections 6 and 7.

Our work opens up intriguing possibilities for interaction between a priori separate research areas. For instance, the web monoid [45], mentioned in Section 7.2.2, is used to recast a version of Baez and Dolan's construction of opetopes [4] (see also [26]). The full generality of our linear algebraic framework (Section 6.2) may be useful in porting this approach to new applications. Our perspective on the structure of opetopes (Section 7.2.2) provides an inductively defined construction for them that may be amenable to formalisation in proof assistants. At any rate, it has already led to a unification of the algebraic aspects of the theory of abstract syntax and the theory of opetopes [12].

Acknowledgements. We are grateful to Zawadowski for bringing his work with Szawiel [45] to our attention, and both of them for discussions about it. We are also grateful to Kammar for his note [22] and to Rivas for comments. We would like to thank the anonymous reviewers for their helpful suggestions.

References

1 J. Adámek. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae*, 015(4):589–602, 1974.

2 S. Alves, M. Fernández, M. Florido, and I. Mackie. Linear recursive functions. In *Rewriting, Computation and Proof: Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of His 60th Birthday*, pages 182–195. Springer, 2007.

3 R. Backhouse, M. Bijsterveld, R. van Geldrop, and J. van der Woude. Categorical fixed point calculus. In *Category Theory and Computer Science, CTCS 1995*, volume 953 of *Lecture Notes in Computer Science*, pages 159–179. Springer, 1995.

4 J. C. Baez and J. Dolan. Higher-dimensional algebra III. n-categories and the algebra of opetopes. *Advances in Mathematics*, 135(2):145–206, 1998. doi:10.1006/aima.1997.1695.

5 J. M. Boardman and R. M. Vogt. Homotopy-everything H-spaces. *Bull. Amer. Math. Soc.*, 74(6):1117–1122, 11 1968.

6 A. Burroni. Récursivité graphique (1^e partie) : catégorie des fonctions récursives primitives formelles. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 27(1):49–79, 1986.

7 J. R. B. Cockett. List-arithmetic distributive categories: Locoi. *Journal of Pure and Applied Algebra*, 66(1):1–29, 1990. doi:10.1016/0022-4049(90)90121-W.

8 P. M. Cohn. *Universal algebra*. Reidel, 2nd edition, 1981.

9 M. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. Cambridge University Press, 1996.

10 M. Fiore. Second-order and dependently-sorted abstract syntax. In *Proceedings of the 2008 23rd Annual IEEE Symposium on Logic in Computer Science*, LICS'08, pages 57–68. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.38.

11 M. Fiore. An equational metalogic for monadic equational systems. *Theory and Applications of Categories*, 27(18):464–492, 2013.

12 M. Fiore. An algebraic combinatorial approach to opetopic structure. Notes and talk, *Seminar on Higher Structures*, Program on Higher Structures in Geometry and Physics, Max Planck Institute for Mathematics, February–March 2016.

13 M. Fiore and C.-K. Hur. On the construction of free algebras for equational systems. *Theoretical Computer Science*, 410(18):1704–1729, 2009. doi:10.1016/j.tcs.2008.12.052.

14 M. Fiore and C.-K. Hur. Second-order equational logic. In *Computer Science Logic, CSL 2010*, volume 6247 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2010.

15 M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, LICS'99. IEEE Computer Society, 1999.

16 G. Gonzalez. The list-transformer package, 2016. URL: https://hackage.haskell.org/package/list-transformer.

17 M. Hamana. Term rewriting with variable binding: An initial algebra approach. In *Proceedings of the 5th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, PPDP'03, pages 148–159. ACM, 2003.

18 A. Hirschowitz and M. Maggesi. Modules over monads and linearity. In *WoLLIC 2007*, number 4576 in Lecture Notes in Computer Science, pages 218–237. Springer, 2005.

19 M. Jaskelioff. *Lifting of Operations in Modular Monadic Semantics*. PhD thesis, University of Nottingham, 2009.

20 A. Joyal. Foncteurs analytiques et espèces de structures. In *Combinatoire Énumérative*, volume 1234 of *Lecture Notes in Mathematics*, pages 126–159. Springer, 1986.

21 A. Joyal. The Gödel incompleteness theorem, a categorical approach (abstract). *Cah. Top. Géom. Diff. Cat.*, 16(3), 2005. Short abstract of the talk given at the International conference *Charles Ehresmann: 100 ans*, Amiens, 7–9 October, 2005.

22 O. Kammar. Algebraic construction of the list transformer. Private communication, 2014.

23 G. M. Kelly. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on. *Bulletin of the Australian Mathematical Society*, 22(1):1–83, 1980. doi:10.1017/S0004972700006353.

24 A. Kock. Bilinearity and cartesian closed monads. *Mathematica Scandinavica*, 29(2):161–174, 1971.

25 A. Kock. Strong functors and monoidal monads. *Archiv der Mathematik*, 23(1):113–120, 1972. doi:10.1007/BF01304852.

26 J. Kock, A. Joyal, M. Batanin, and J.-F. Mascari. Polynomial functors and opetopes. *Advances in Mathematics*, 224(6):2690–2737, 2010. doi:10.1016/j.aim.2010.02.012.

27 J. Lambek. Multicategories revisited. In *Categories in Computer Science and Logic: Proc. of the Joint Summer Research Conference*, pages 217–239. American Mathematical Society, 1989.

28 F. W. Lawvere. Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences of the United States of America*, 50(5):869–872, 1963.

29 F. W. Lawvere. An elementary theory of the category of sets. *Proceedings of the National Academy of Sciences of the United States of America*, 52(6):1506–1511, 1964.

30 D. J. Lehmann and M. B. Smyth. Algebraic specification of data types: A synthetic approach. *Mathematical Systems Theory*, 14(1):97–139, 1981. doi:10.1007/BF01752392.

31 T. Leinster. *Higher Operads, Higher Categories*. Number 298 in London Mathematical Society Lecture Note Series. Cambridge University Press, 2004.

32 S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, 2nd edition, 1998.

33 M. E. Maietti. Joyal's arithmetic universe as list-arithmetic pretopos. *Theory and Applications of Categories*, 24(3):39–83, 2010.

34 M. Markl. Operads and PROPs. In *Handbook of Algebra, Volume 5*, pages 87–140. Elsevier, 2008.

35 P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.

36 J. P. May. *The Geometry of Iterated Loop Spaces*, volume 271 of *Lecture Notes in Mathematics*. Springer, 1972.

37 B. Nordström, K. Peterson, and J. Smith. *Programming in Martin Löf's Type Theory*. Clarendon Press, 1990.

38 R. Paré and L. Román. Monoidal categories with Natural Numbers Object. *Studia Logica*, 48(3):361–376, Sep 1989. doi:10.1007/BF00370829.

39 Maciej Piróg. Eilenberg-Moore monoids and backtracking monad transformers. In *Proceedings 6th Workshop on Mathematically Structured Functional Programming, MSFP@ETAPS 2016*, pages 23–56, 2016. doi:10.4204/EPTCS.207.2.

40 B. Plotkin. *Universal Algebra, Algebraic Logic, and Databases*. Springer, 1994.

41 J. Power. Enriched Lawvere theories. *Theory and Applications of Categories*, 6:83–93, 1999.

42 E. Rivas, M. Jaskelioff, and T. Schrijvers. From monoids to near-semirings: The essence of MonadPlus and Alternative. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming*. ACM, 2015.

43 R. M. Robinson. Primitive recursive functions. *Bull. Amer. Math. Soc.*, 53(10):925–942, 10 1947.

44 J. M. Spivey. Algebras for combinatorial search. *J. Funct. Program.*, 19(3-4):469–487, 2009. doi:10.1017/S0956796809007321.

45 S. Szawiel and M. Zawadowski. The web monoid and opetopic sets. *Journal of Pure and Applied Algebra*, 217:1105–1140, 2013. doi:10.1016/j.jpaa.2012.09.030.

46 T. Uustalu. A divertimento on MonadPlus and nondeterminism. *Journal of Logical and Algebraic Methods in Programming*, 85(5):1086–1094, 2016. doi:10.1016/j.jlamp.2016.06.004.

47 W. G. Van Hoorn and B. Van Rootselaar. Fundamental notions in the theory of seminearrings. *Compositio Mathematica*, 18(1-2):65–78, 1967.

48 H. Wolff. Monads and monoids on symmetric monoidal closed categories. *Archiv der Mathematik*, 24(1):113–120, 1973. doi:10.1007/BF01228184.

Is the Optimal Implementation Inefficient? Elementarily Not*

Stefano Guerrini[1] and Marco Solieri[†2]

1 **LIPN, Université Paris 13 – Sorbonne Paris Cité, Paris, France**
`stefano.guerrini@univ-paris13.fr`
2 **Department of Computer Science, University of Bath, Bath, UK**
`ms@xt3.it`

Abstract

Sharing graphs are a local and asynchronous implementation of lambda-calculus beta-reduction (or linear logic proof-net cut-elimination) that avoids useless duplications. Empirical benchmarks suggest that they are one of the most efficient machineries, when one wants to fully exploit the higher-order features of lambda-calculus. However, we still lack confirming grounds with theoretical solidity to dispel uncertainties about the adoption of sharing graphs. Aiming at analysing in detail the worst-case overhead cost of sharing operators, we restrict to the case of elementary and light linear logic, two subsystems with bounded computational complexity of multiplicative exponential linear logic. In these two cases, the bookkeeping component is unnecessary, and sharing graphs are simplified to the so-called "abstract algorithm". By a modular cost comparison over a syntactical simulation, we prove that the overhead of shared reductions is quadratically bounded to cost of the naive implementation, i.e. proof-net reduction. This result generalises and strengthens a previous complexity result, and implies that the price of sharing is negligible, if compared to the obtainable benefits on reductions requiring a large amount of duplication.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic

Keywords and phrases optimality, sharing graphs, λ-calculus, complexity, linear logic

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.17

1 Introduction

1.1 Intelligence of sharing graphs

Redundancy and non-locality. An ideal implementation of a functional programming language aims at satisfying two properties: sharing, i.e. to avoid the duplication of work, and locality, i.e. to be parallelisable on architectures with multiple computing agents. Although these properties are not orthogonal, a prerequisite for both is a fine-grained implementation of material duplication. Consider the λ-calculus and take for instance the β-redex $T = (\lambda x.M)(\lambda y.N)$ and assume that x occurs $k+1$ times in M. In the λ-term $M\{^{\lambda y.N}/_{x}\}$ obtained by reducing it, we have k new copies of N, and then, k additional copies of any redex in it. Also, such a reduction step cannot be fired in parallel with any reduction in N. To solve this kind of problems, we may consider switching to graph reduction. At the time

* Preliminary results of this work were previously presented [14] with a stronger yet unsolved claim.
† Work mainly carried out during: PhD studentship at Paris 13 and Bologna, ATER at Paris 7.

of writing, the technique is employed for instance in the Glasgow Haskell Compiler, namely by the STG-machine at its core [21], whilst its essence dates back to the early seventies [24]. The key idea can be formalised with proof-nets of linear logic (LL), where T can be expressed as in Figure 1a. With respect to the syntax tree of T we notice: the explicit connection of the λ-link with its variable x; a ?-link connecting the $k+1$ occurrences of x in M; a box (the dotted square) around the argument of the application (the @-link), to mean $\lambda y.N$ is a duplicable term. Such information can be equivalently formalised by associating the boxing depth to each link – e.g. the rightmost λ-link has index 1. Now, a linear-β reduction step rewrites Figure 1a to Figure 1b. The box is now shared, thus reductions in N can be performed without duplicating it. Unfortunately, this is still quite unsatisfactory. Indeed, if an occurrence of x, which is represented in the figure as a premiss of the topmost contraction ?-link, appears in argument position within M, i.e. it is linked to a @-link, then we have a "virtual β-redex". In particular, this @-link and the λ-link will form a redex, once we reduce the exponential redex between the ?-link and the promotion !-link. The problem is that firing the latter entails the duplication of the whole box, and thus the loss of sharing benefits.

Sharing and locality. Sharing graphs [19, 13, 4] include instead, by their very design, the solution to these problems of duplication. Back to our example, instead of performing the duplication of the lowermost box, we can apply the "triggering" rule (t) shown in Figure 1b-c. This introduces a link of a new kind $|\!+\!\rangle$, called *mux* (multiplexer), that has $k+1$ premisses and index 1 as the content of the box. Muxes perform duplications of boxes in a local, link-by-link way, following the approach introduced by interaction nets [18]. For instance, the $|\!+\!\rangle$-link in the example can duplicate the λ-link only, by the $(d^{\pm}\!\circ)$ rule in Figure 1c-d. This allows N being kept shared, whilst copies of the λ-link can now independently interact with @-links in M. Also, from the $(d^{\pm}\!\circ)$ rule originates a sort of co-sharing link, the *negative mux* with kind $|\!-\!\rangle$ and the same index, which implements sharing of contexts (outputs), instead of terms (inputs). In addition to the case of λ-links, there is one duplication for each other link $(@, !, ?)$, but it is applicable only when the mux faces its principal port, i.e. its conclusion, depicted by an outgoing arrow. This makes duplication *lazy* – it is performed only when a mux obstructs the formation of other redexes. The life of a $|\!+\!\rangle$-link may end in two ways: by being merged into it, when it reaches the premiss of a ?-link, or by annihilating with a facing $|\!-\!\rangle$-links (i.e. the facing pair of links reduces to the identity). But positive muxes may also need to swap with negative ones, i.e. they duplicate each other. Therefore, we mark muxes with the index inherited from the exponential redex, e.g. 1 in our running example, so we can distinguish two kinds of redexes with two opposite muxes: annihilation when indices are equal, swap when they are different. In general, we would also need a supplemental mechanism or information tracking, the so-called "oracle", which manages indexes, as a local implementation of digging and dereliction of LL. But this is not the case for proof-nets for terms typed in the elementary and light variants of LL (ELL and LLL) [12, 6]. They are obtained by a restriction on usual exponential boxes that makes indices immutable by definition. Hence, for their sharing implementation we can simply consider the so-called *abstract algorithm* of sharing graphs (ASG). Thanks to muxes, the sharing graph G that is the normal form of T is a (possibly enormously) compressed representation of the proof-net $\overline{T}$ we would have obtained by ordinary cut-elimination. To retrieve $\overline{T}$ from G, we need the *read-back* (RB) – a set of additional rewriting rules for sharing graphs, which unlatch muxes to let them freely duplicate downwardly, i.e. from the root of the term to its inputs.

▷ This paper considers elementary proof-nets (EPN) to represent ELL and LLL typed terms, or proofs (presented in Section 2), and their sharing implementation with ASG, including RB (Section 3).

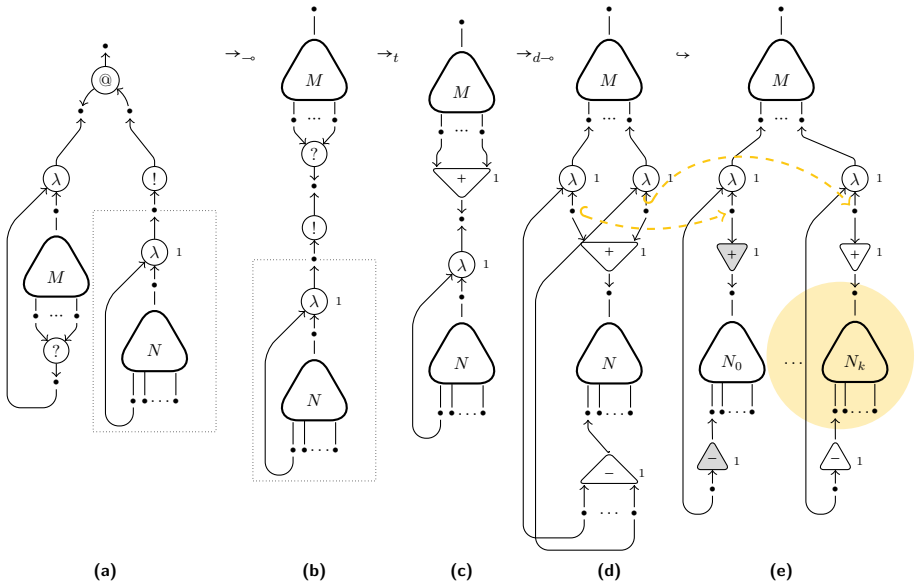


Figure 1 Examples: sharing reductions (a-d) on a proof-net; unshared graph (e) unfolding of d.

1.2 Efficiency of sharing graphs

Question. The possible benefits of sharing were astonishingly evident from the very first sequential implementations of sharing graphs. For example, in the normalisation of λ-terms benchmarks of BOHM [3] recorded polynomial times, against traditional languages (Caml Light and Haskell) [4, Ch. 10] requiring exponential times. But sharing and locality may come with a price. *What is the price of sharing graphs?* To answer this question, very recently broadly surveyed by Asperti [1], we first need the notion of cost. We cannot use the number of β steps, since, even though they are a reasonable [23] measure for the λ calculus [9], in sharing graphs whole families of redexes are reduced simultaneously, (and this is the reason why they realise the Lévy optimal reduction [20]). Nor can we use the number of such parallel β-steps, since it would be an enormously parsimonious measure. Indeed, a polynomial number of family reductions in the size s of a term may hide a concrete cost bigger than a tower of exponentials of s, caused by the oracle rules [5] or by the local duplication rules [2]. Therefore, a study of the complexity of sharing graphs is necessarily limited by the state of the art to take the form of a comparison to some existing reduction system, i.e. by the approach of "ICC in the small" [10], and to use a cost measure based on standard rewriting theory – counting the size of sub-graphs that are erased/introduced at each step. Hence, to employ a reasonable measure we are possibly renouncing to parsimoniousness.

▷ In Section 4.1 we shall define two cost functions for ASG and EPN reductions.

A first partial answer. In the only previous contribution which tackled the complexity study of ASG [7], Baillot, Coppola and Dal Lago exploited the implicit computational complexity of (the affine variants of) ELL and LLL. The cost of the cut-elimination of a proof-net N with maximum boxing index d is related respectively by a Kalmar elementary function in the size of N and rank d, in ELL, or by a polynomial function in the size of N and degree d, in LLL. By means of a quantitative semantical tool [8] inspired by the geometry of interaction [11], the authors proved that the cost of a normalisation with sharing graphs of ELL and LLL proof-nets remains in the two aforementioned complexity classes. But they were not able to

give any explicit bound to the overhead that sharing graphs might introduce in the worst case – for instance, when it becomes less effective, since the reduction does not require a relevant amount of duplications. Moreover, the technical approach hardly seems adaptable to prove a similar result for the more general case featuring the "oracle".

Contributions. We study the complexity of reducing (proof-nets representing) λ-terms typed in the elementary or light linear logic and relate it to the cost of naive reduction. In particular, we give a worst-case bound of the cost of sharing reduction as a quadratic function in the cost of the naive reduction. Three are the axis along which this paper improves the previous literature [7]:

1. *Strength.* We give a clear bound to the overhead of sharing reduction.
2. *Generality.* Our analysis considers strategy-agnostic reductions of arbitrary length, instead of normalisation, and includes the read-back rules (which are sub-optimal). In this way we get a more uniform, and perhaps fairest, complexity comparison.
3. *Scalability.* The technical approach bases on a quantitative extension of an elegant syntactical simulation between sharing graphs and proof-nets; this provides modularity for our complexity analysis, and appears to give room for further investigations also about more general cases.

▷ The complexity bound is obtained in Sections 4.2 and 4.3. Detailed proofs are omitted for lack of space, but can be found in an extended version available on authors' websites.

2 Intuitionistic elementary and light logics

We start by introducing proof-nets of ELL and LLL. We define first the two intuitionistic and weakening-free logics by a levelled sequent calculus inspired by the approach of Guerrini, Martini and Masini [15], which represents a typing system for λI terms. The absence of weakening in λI will remove a considerable technical overhead that have no impact on the complexity question (in the literature [7, for instance] garbage collection is indeed quite commonly postponed at the end) nor on the proof technique. On the other hand, this will greatly ease the exposition of ASG and their complexity analysis, since it removes reduction rules and entails the uniqueness of the root in sharing graphs.

Then, we give the translation of sequent proofs in levelled proof-nets, that are directed hypergraphs where every (occurrence of a) formula is a vertex and every inference rule is a directed hyperedge (called link) that goes from the premisses of the rule to its conclusions. Axioms and cuts correspond instead to a direct plugging of the two sub-proofs. Finally, we will present cut-elimination as reduction of proof-nets.

2.1 Sequent calculus and typed lambda terms

▶ **Definition 1** (Logics and typing)**.** Given L a set of *literal* symbols, the *formulas* of ELL and LLL are built from the following grammar.

$$T ::= L \mid T \multimap T \mid !T \mid \S T. \tag{1}$$

A *levelled formula* is a pair T^n where T is a formula and $n \in \mathbb{N}$. Given a set V of *variables*, the set of *terms* is defined from the standard definition of the Λ calculus:

$$t ::= V \mid \lambda V.t \mid t\,t, \tag{2}$$

$$\frac{}{x : A^n \vdash x : A^n}\ (\mathrm{Ax}) \qquad \frac{\Delta_1^{n+1}, \Gamma_1^n \vdash t : A^n \qquad \Delta_2^{n+1}, \Gamma_2^n, x : A^n \vdash u : B^n}{\Delta_1^{n+1}, \Delta_2^{n+1}, \Gamma_1^n, \Gamma_2^n \vdash u[t/_x] : B^n}\ (\mathrm{Cut})$$

$$\frac{\Delta^{n+1}, \Gamma^n, \{x : A^n\} \vdash t : B^n}{\Delta^{n+1}, \Gamma^n \vdash \lambda x.t : A \multimap B^n}\ (\multimap) \qquad \frac{\Delta_1^{n+1}, \Gamma_1^n \vdash t : A^n \qquad \Delta_2^{n+1}, \Gamma_2^n, x : B^n \vdash u : C^n}{\Delta_1^{n+1}, \Delta_2^{n+1}, \Gamma_1^n, \Gamma_2^n, y : A \multimap B^n \vdash u[yt/_x] : C^n}\ (\bar{\multimap})$$

(a) Common rules: axiom, cut, abstraction, application

$$\frac{\Gamma^{n+1} \vdash t : A^{n+1}}{\Gamma^{n+1} \vdash t : !A^n}\ (!) \qquad \frac{\{x : A^{n+1}\} \vdash t : A^{n+1}}{\{x : A^{n+1}\} \vdash t : !A^n}\ (!) \qquad \frac{\Gamma^{n+1}, \Delta^{n+1} \vdash t : A^{n+1}}{\S\Gamma^n, \Delta^{n+1} \vdash t : \S A^n}\ (\S)$$

(b) Elementary promotion

(c) Light promotion – Γ, Δ may be empty

$$\frac{\Delta^{n+1}, \Gamma^n, (x_i : A^{n+1})_{1 \le i \le m} \vdash t : B^n}{\Delta^{n+1}, \Gamma^n, y : !A^n \vdash t[y/_{x_i}]_{1 \le i \le m} : B^n}\ (?)$$

(d) Contraction

Figure 2 Levelled sequent calculus of ELL and LLL.

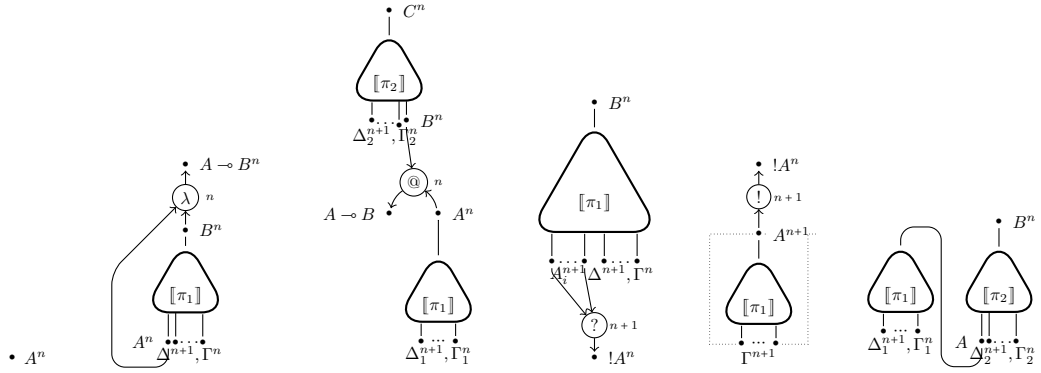
by constraining abstraction so that it bounds at least one occurrence (i.e. x must appear as a free variable of t in order to write $\lambda x.t$). The *sequent calculus* of the *first-order, weakening-free, intuitionistic fragment of ELL and LLL* is defined in Figure 2 and represents a type assignment system for the Λ_I calculus.

▶ Remark 2 (LLL ⊂ ELL). Observe that § cannot appear in ELL formulas and that the light version of the (!) rule is an instance of the elementary one. Moreover, any proof π of LLL can be encoded in ELL by using the logical modality ! instead of §. For this reason and the sake of simplicity, in the rest of the paper we shall refer only to the elementary version.

2.2 Proof-nets

▶ **Definition 3** (Elementary proof-nets)**.** Given a set of *vertices*, a *link* is a directed hyperedge, i.e. a triple $(V\ \kappa^n\ u)$, where: u is a vertex called *conclusion*, V is a non empty sequence of vertices called *premisses*, κ is a label called *kind*, $n \in \mathbb{N}$ is called *level*. Depending on the kind of a link l, a *polarity* is assigned to each of its vertices, i.e. an element of $\{\iota, o\}$ (input and output), and the *arity* may be fixed, i.e. the number of vertices of l. In order to keep a better correspondence with the underlying λ-calculus, the kinds of links will correspond to the syntactical construct associated to the rule and not to the corresponding logical connective. Also, links are depicted so that positioning follows the input/output computational interpretation, o above, ι below; whilst arrows orientation follows the proof-theoretical viewpoint, premisses inward, conclusions outward.

A proof-net of the fragment of ELL of our interest, to which we shall simply refer to as an *elementary proof-net* (EPN), is the hyper-graph N obtained by the *translation* $[\![\pi]\!]$ of some sequent proof π, as inductively defined as follows. Let R be the last rule of π, assume its shape to be as in one of Figures 2a, 2b, and 2d. Also, let π_1, π_2 respectively be the sub-proofs of π (if any) whose conclusions are the leftmost and rightmost premisses of R. Then the translation of π is given by the case analysis in Figure 3. There we assume that $[\![\pi_1]\!], [\![\pi_2]\!]$ are disjoint, and also that, except when otherwise depicted, both are disjoint from the vertices introduced by the inductive steps (i.e. new vertices are "fresh"). Although we shall label vertices with their names, in the picture we used formulas to ease the reading and stress the correspondence with the sequent proof.



(a) Axiom **(b)** Abstraction **(c)** Application **(d)** Contraction **(e)** Promotion **(f)** Cut

Figure 3 Elementary proof-nets, as translated from sequent proofs. See Definition 3 and Figure 2.

The *conclusions* of $[\![\pi]\!]$ are the vertices corresponding to formulas appearing in the sequent proved by π – *input* vertices stand for formulas on the left-hand side of the sequent, *output* ones for those on the right-hand side. The set of the conclusions of $N \in \mathsf{EPN}$ is called its *interface* and denoted by $\mathsf{iface}(N)$; any vertex of N not in $\mathsf{iface}(N)$ is an *internal vertex* of N, and the corresponding set is denoted by $\mathsf{int}(G)$.

▶ Remark 4 (λ-terms). The input-output relation allows to associate a λ-term to a proof-net – if we erase exponential links, we essentially obtain the syntax tree of the term.

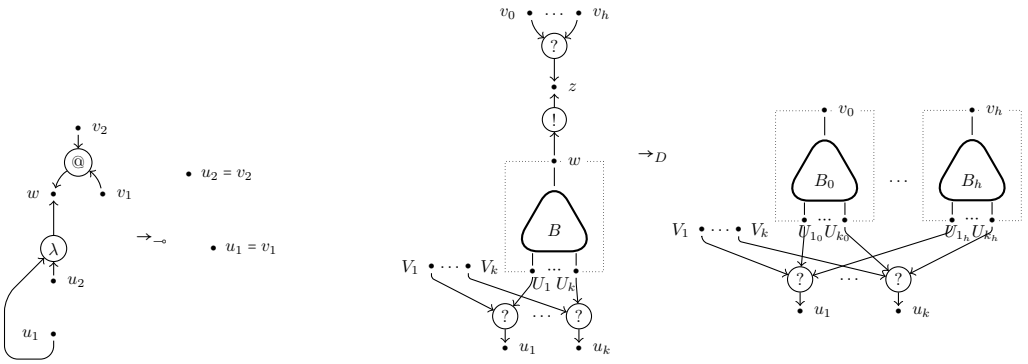
▶ **Definition 5** (Boxes)**.** The level of a vertex v is denoted as $\ell(v)$, and we shall write the same for a link, meaning the level of its premiss(es). The maximum level of links in N is written $\partial(N)$. A *box* of level n in $N \in \mathsf{EPN}$ is a sub-graph B of N whose links and vertices have level not smaller that n and that is maximal with respect to inclusion. The ι conclusion v of B is called its *principal door*, whilst the o conclusions $u_1, \dots, u_k$ *auxiliary doors*. We shall then denote B as $(u_1, \dots, u_k \ \langle\!\langle B \rangle\!\rangle \ v)$. Boxes are depicted with dotted squares.

▶ **Definition 6** (Paths)**.** A *path* from u_0 to u_k in a graph G is a sequence of vertices $(u_0, \dots, u_k)$ for which there is a sequence of links $l_0, \dots, l_{k-1}$ such that $u_i \neq u_{i+1}$ and u_i belongs to both the vertices of l_i and those of l_{i+1}, for any $0 \leq i < k-1$. A *downward path* is a path such that: u_i is not the first premiss of a λ-link and is an o-vertex of l_{i+1}; and u_{i+1} is an ι-vertex of l_{i+1}, for any $0 \leq i < k-1$. If these holds, then $(u_k, \dots, u_0)$ is an *upward path* from u_k to u_0. We shall write $u \sim v$ when there is a path from u to v, $u \rightsquigarrow v$ when there is a downward one, $v \rightsquigarrow u$ when $u \rightsquigarrow v$. A *rooted* path is a downward path starting from the o-conclusion of G.

▶ Remark 7 (Boxes). Auxiliary doors of a box B are always premisses of an exponential link, and that B is always connected: there exists a path between any two of its vertices. Also, boxes properly nest: given two distinct boxes, either they are completely disjoint, or one is included into the other. Our definition of box is slightly different from the standard one, as it is does not include vertices with exponential formulas, but just take the interior of the box.

▶ **Definition 8** (EPN reduction)**.** The rewriting relation $\to_{\mathsf{EPN}}$, which implements the cut-elimination on proof-nets, is obtained by the context closure of the reduction rules $(\overset{+}{\multimap}), (D)$ respectively defined in Figures 4a and 4b.

▶ **Notation 9.** Reductions steps are denoted by Greek letters (e.g. ρ), sequences are marked with overlines ($\overline{\rho}$), reducts are denoted by functional notations ($N \to \rho(N)$).



(a) Multiplicative **(b)** Exponential

Figure 4 EPN reduction.

▶ **Proposition 10** (Stratification). *In EPN, levels are preserved by reduction.*

3 Abstract sharing graphs

We introduce abstract sharing graphs (ASG) and their reduction, and recall their most important qualitative properties as an implementation of EPN.

3.1 Syntax and computation

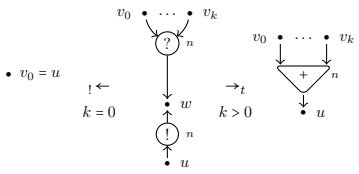
▶ **Definition 11** (Sharing and read-back reductions). The *sharing reduction* is the graph-rewriting relation $\to_{\mathsf{ASG}}$ given by the context closure of the following reduction rules.
Logical $(\multimap), (!), (t)$, defined in Figures 4a and 5a. On unary contractions, $(!) = (D)$.
Duplication $(d\overset{+}{\multimap}), (d\overset{-}{\multimap}), (d!), (d?)$, respectively defined in Figures 5c, 5d, 5e, and 5f.
Muxes $(a), (s)$, defined in Figure 5b.
The *read-back reduction* $\to_{\mathsf{RB}}$ is obtained from the mux interaction rules and the followings.
Read-back duplication $(r\overset{+}{\multimap}), (r?), (m)$, defined in Figures 5g and 5h.
The RB-normal form of a graph G is called its *read-back* and written $\mathcal{R}(G)$. The reduction $\to_{\mathsf{ASGR}}$ is the union of $\to_{\mathsf{ASG}}$ and $\to_{\mathsf{EPN}}$. The set ASG of abstract sharing graphs is obtained by the closure of EPN with respect to $\to_{\mathsf{ASGR}}$.

3.2 Implementation of EPN

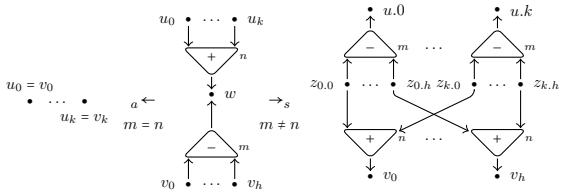
Sharing graphs with ASG and RB reductions represent a well-behaved rewriting system. $\to_{\mathsf{ASG}}$ is locally confluent. $\to_{\mathsf{RB}}$ and $\to_{\mathsf{ASGR}}$ are confluent. All three are strongly normalising, and the last two have normal forms in EPN [16, Thm.s 4, 11.i and 11.ii (for MELL)]. The traditional way of normalising proofs or terms with sharing graphs maximises the amount of sharing by postponing duplication as much as possible, thus performing first an ASG-normalisation and then an RB one. This gives a correct implementation of EPN reduction and a complete implementation of EPN normalisation.

▶ **Theorem 12** (Correctness). *If $N \in EPN, G \in ASG$ and $N \to^*_{ASG} G$, then $N \to^*_{EPN} \mathcal{R}(G)$.*

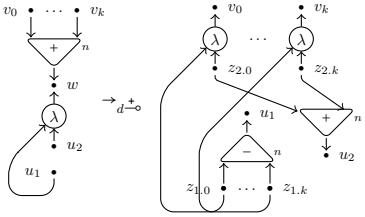
Proof. We refer to the original proof for λ-calculus [13], or the more syntactic one for MELL [16, Thm. 13]. ◀



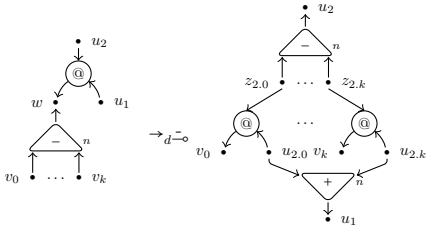
(a) Exponential reductions



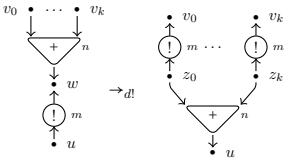
(b) Annihilation and swap



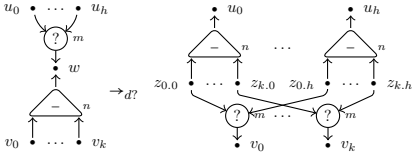
(c) Duplication of abstraction



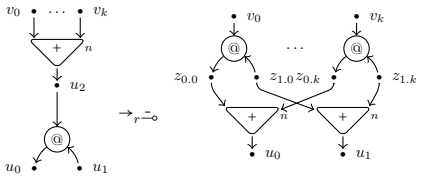
(d) Duplication of application



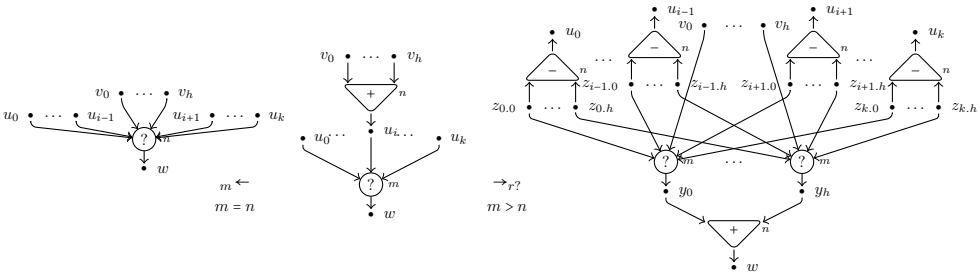
(e) Duplication of promotion



(f) Duplication of contraction



(g) Duplication of application



(h) Merge with, or duplication of, contraction

Figure 5 ASG and RB reduction rules.

▶ **Theorem 13** (Normalisation completeness). *If $N, \bar{N} \in EPN$ where $N \to^*_{EPN} \bar{N}$, with $\bar{N}$ normal, then there is $\bar{G} \in ASG$ being ASG-normal and such that $N \to^*_{ASG} \bar{G} \to^*_{RB} \bar{N}$.*

Proof. See Asperti and Guerrini [4, Thm. 7.9.3.ii]. ◀

If instead we consider $\to_{\mathsf{ASGR}}$, sharing graphs can be shown to be more generally complete with respect to the whole EPN-reduction (not just normalisation). It suffices to prioritise RB redexes, thus enforcing exhaustive duplications of boxes.

▶ **Theorem 14** (Completeness). *For any $N, N' \in EPN$ if $N \to_{EPN} N'$ then $N \to^+_{ASGR} N'$.*

▶ Remark 15 (Optimality). Once equipped with a "call-by-need" strategy [4, §5.6], the number of $(-\circ)$ steps performed by ASG is minimised so the reduction reaches Lévy-optimality [16, Thm. 14 (for MELL)] [4, Thm. 5.6.4 (for λ-calculus)]. For the sake of generality, our focus will not be limited to the optimal strategy, and we shall analyse sharing graphs with the greatest strategy-agnosticism.

4 Computational complexity

4.1 Cost measures

We define two cost functions $\mathcal{C}_{\mathsf{ASG}}$ and $\mathcal{C}_{\mathsf{EPN}}$, respectively for the ASG and EPN reductions. Both measures are essentially equivalent to the size of the rewriting operations required. However, to ease the presentation without affecting fairness, only $\mathcal{C}_{\mathsf{EPN}}$ is formally defined as such, whilst $\mathcal{C}_{\mathsf{ASG}}$ has been accurately hand tuned.

▶ **Definition 16** (Size and variations). The *size* of a graph G, written $\#G$, is the sum of the cardinality of the set of its vertices and the sum of the arities of its links. We remark that for a box $(u_1, \dots, u_k \ \langle\!\langle B \rangle\!\rangle \ v)$, all of its doors, principal and auxiliary ones, belong to the sub-graph B and are accounted by $\#B$. Given M a metric on a graph G, e.g. the size of G or of some set of sub-graphs, and ρ a reduction step, $\Delta(\rho)$ denotes $M(\rho(G)) - M(G)$.

▶ **Definition 17** (EPN-reduction cost). The cost $\mathcal{C}_{\mathsf{EPN}}(\rho)$ of a EPN-reduction step ρ on a levelled proof-net N is defined as the size of the symmetric difference between the vertices and links of N and those of $\rho(N)$. Namely, the cost of a given rule is computed in Table 1a. The cost of a reduction sequence $\bar{\rho}$ is the sum of the costs of each step it is composed of.

▶ **Definition 18** (ASG- and RB-reduction costs). The cost $\mathcal{C}_{\mathsf{ASG}}(\sigma)$ and $\mathcal{C}_{\mathsf{RB}}(\sigma)$ of a ASG- or RB-reduction step σ is given in Table 1b.

4.2 Unshared simulation

Our complexity comparison is directly founded on the most natural correspondence between sharing graphs and proof-nets: a *syntactical simulation*. Given a proof-net P, any ASG reduction sequence σ on P can be simulated by a EPN sequence ρ on the same P. Since the only difference between the two rewriting systems is the style of duplication, local or global, respectively, $\sigma(P)$ and $\rho(P)$ may greatly differ in the number of copies of some subgraphs. To understand precisely the relationship between $\sigma(P)$ and $\rho(P)$, we shall employ an intermediate reduction system – the *unshared graphs* (UG) [16]. Let us taste the main intuitions employing the example that we began to consider in Section 1.1 and Figure 1. The key feature of UG is the fact that the exponential redex in Figure 1b would be rewritten in $k+1$ copies of the box (as one usually does on proof-nets), plus $k+1$ unary muxes (instead

Table 1 Costs assigned to classic and sharing reductions.

Rule	$\mathcal{C}_{\mathsf{EPN}}(\rho)$
$(\multimap)$	9
(D)	$k \times \#B + 2k + 4$

(a) Classic reduction rules. In the case of (D), $k+1$ is the number of premisses of the ?-link, and B is the box enclosed by the !-link.

Rule	$\mathcal{C}_{\mathsf{ASG}}(\sigma)$
$(\multimap)$	9
$(!)$	6
(t)	$j+4$
$(d!)$	$3k$
$(d\overset{+}{\multimap}), (d\overset{-}{\multimap}), (r\overset{-}{\multimap})$	$5k$
$(d?), (r?)$	$(2j+3) \times k$
$(a), (m)$	k
(s)	$k \times l$

(b) Sharing and read-back rules. $j+1$, $k+1$ and $l+1$ respectively are the number of premisses of the ?-link, the first and the second $|*)$-link, where involved.

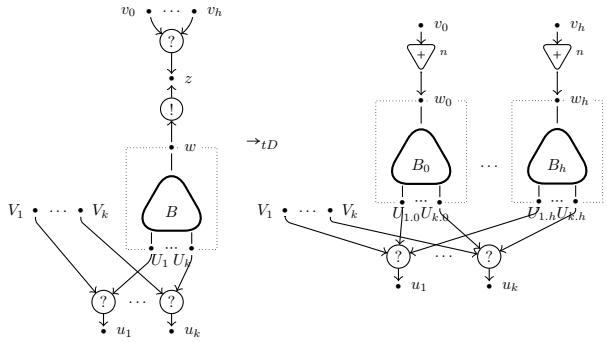


Figure 6 UG reduction rule: duplication and triggering (where $h > 1$).

of one k-ary mux of Figure 1c). Have a peek at Figure 6. The new links are called *lifts*, and play the mere role of markers for the presence of muxes in corresponding sharing graphs. In particular, their propagation along the graph does not affect any other link, and if we erase them we simply obtain a proof-net. For instance, the $(d\lambda)$ step in Figure 1c-d, can be simulated by $k+1$ propagation steps that reach Figure 1e, that is the *unfolding* of Figure 1d. Similarly, any reduction step inside N (which is shared) needs to be performed $k+1$ times in N_i. Yellow arrows illustrate such *unfolding* relation.

4.2.1 Unshared graphs

▶ **Definition 19** (Unshared reduction and graphs)**.** The *unshared reduction* is the rewriting relation $\to_{\mathsf{UG}}$ obtained from $\to_{\mathsf{ASG}}$ by replacing the (t) rule with (tD), defined in Figure 6. The *unshared read-back* $\to_{\mathsf{UR}}$ is the restriction of RB-reduction to UG (i.e. muxes are unary). The union of these two relations is written as UGR. An *unshared graph* UG-graph for short, is either a levelled proof-net, or the reduct of an unshared graph via UG- or UR-reduction.

4.2.2 Unfolding and simulating sharing graphs in unshared graphs

▶ **Definition 20** (Sharing morphism)**.** A *sharing morphism* $\mathcal{M}$ is a surjective homomorphism on hypergraphs [17] from UG to ASG that preserves the kind and level of links. We say that $G \in \mathsf{ASG}$ *unfolds* to $U \in \mathsf{UG}$, written $G \hookrightarrow U$, if there is a sharing morphism $\mathcal{M}$ such that $\mathcal{M}(U) = G$. We shall use the same notation to relate vertices and links: if $v \in V(G)$ and

$W \subseteq V(U)$ we write $v \hookrightarrow W$ to mean $\mathcal{M}(W) = v$, while if $m \in L(G)$ and $N \subseteq L(U)$ we write $m \hookrightarrow N$ when $\mathcal{M}(N) = m$.

▶ **Lemma 21** (Unfolded simulation). *For any $N \in \mathsf{EPN}$, $G \in \mathsf{ASG}$, if $N \xrightarrow{\bar{\sigma}}{}^{*}_{\mathsf{ASG}} G$ then there exists $U \in \mathsf{UG}$ such that $N \xrightarrow{\bar{\mu}}{}^{*}_{\mathsf{UG}} U$ and $G \hookrightarrow U$. Moreover, for any $G' \in \mathsf{ASG}$, if $G \xrightarrow{\bar{\sigma}'}{}^{*}_{\mathsf{RB}} G'$ then there exists $U' \in \mathsf{UG}$ such that $U \xrightarrow{\bar{\mu}'}{}^{*}_{\mathsf{UR}} U'$ and $G' \hookrightarrow U'$. We call $\bar{\mu}$ and $\bar{\mu}'$ an* unfolded simulation *of $\bar{\sigma}$ and $\bar{\sigma}'$, respectively, and write $\bar{\sigma} \hookrightarrow \bar{\mu}$ and $\bar{\sigma}' \hookrightarrow \bar{\mu}'$.*

4.2.3 Simulating sharing graphs into proof-nets

▶ **Definition 22** (Lift erasure). The *lift erasure* is the function $\mapsto$ that maps a $U \in \mathsf{UG}$ to the $N \in \mathsf{EPN}$ obtained by equating any two vertices u, v for which there is $(u \mid\!\ast\!\rangle\ v) \in L(U)$. The function is extended to let it map vertices and links of U to those of N.

▶ **Lemma 23** (Lift erasure simulation). *For any $N \in \mathsf{EPN}, U \in \mathsf{UG}$, if $N \to^{*}_{\mathsf{UG}} U$ then there is a unique $\bar{\sigma} : N \to^{*}_{\mathsf{EPN}} N'$ such that $U \mapsto N'$.*

▶ **Definition 24** (Sharing implementation). Given $N \in \mathsf{EPN}, G \in \mathsf{ASG}$, we say N is *implemented by* G, written $G \twoheadrightarrow N$, if there is $U \in \mathsf{UG}$ such that $G \hookrightarrow U \mapsto N$.

▶ **Theorem 25** (EPN-reduction simulates ASG-reduction). *Let N be a proof-net. If $N \to^{*}_{\mathsf{ASG}} G$ then there exists $N \to^{*}_{\mathsf{EPN}} N'$ such that $G \twoheadrightarrow N'$.*

4.3 Quantitative unshared simulation

Given two simulating reduction sequences on the same proof-net, σ of ASG and ρ of EPN, we can compare $\mathcal{C}_{\mathsf{ASG}}(\sigma)$ to $\mathcal{C}_{\mathsf{EPN}}(\rho)$, finding a quadratic bound. In order to do so, we shall bring the unshared simulation up to a quantitative level, so that these two costs can be expressed as two different cost measures on unshared reduction, thus allowing a direct comparison.

4.3.1 Share

The first step is to define introduce a labelled reduction for UG which decorates lifts: at the creation of a set of lifts we add to each of them a fresh label tracking their number and relative indices. We interpret the 0-th lift as the master one, while all the others as sharing ones. In the previous illustration in Figure 1e, we darkened master lifts. Then, given an unshared graph U being the unfolding of a sharing graph G, we can build the *sharing context* of a vertex v as the sequence of lift labels that are along its access path from the root of U. Its sharing context allows us to understand:

1. if v is shared in G, i.e. if its context contains non master lifts (equivalently: "has v been previously copied in U"?);
2. how much v is shared, i.e. the product of the number of lifts in the label, for any lift in the context of v ("how many other copies of v are in U"?).

The set of such shared objects is called the *share*, and conceptually represents the subtraction of the graph of G from that of U. In Figure 1e, we highlighted the share as a pale yellow circle. Notice that it does not include N_0, since its sharing context include a master lift, we interpret it as a *master* copy of N (the stereotype).

▶ **Definition 26** (Copy identity). We enrich $\to_{\mathsf{UGR}}$ with the *copy identity* labelling (CID). It maps lifts to *labels* of the form $x_{i:k}$ (positive) or $\overline{x}_{i:k}$ (negative), where $x \in \mathcal{V}$ a set of variable symbols, whilst $i, k \in \mathbb{N}$ are the *current* and *maximal index*. Given $U \in \mathsf{UG}$ and μ a (tD)-step

as in Figure 6, we set $\mathsf{CID}((v_j \mid\!*\!\rangle^n u_j)) = x_{j:h}$, for any $0 \le j \le h$, and for some $x \in \mathcal{V}$ not occurring in the labels of U. Labels are negated in negative lifts: if l is a positive lift with $\mathsf{CID}(l) = x_{j:h}$ and l' is a negative residual of l w.r.t. a lift propagation rule $(d\kappa)$ or $(r\kappa)$, then $\mathsf{CID}(l') = \overline{x}_{j:h}$. In any other case, labels are preserved by reduction, in particular under copying.

▶ **Definition 27** (Sharing contexts)**.** The *sharing contexts* $\mathfrak{C}$ are the strings generated by the binary concatenation operator $\cdot$ over labels, and including 1 as the identity element (i.e. the empty string) and 0 as the absorbing element for concatenation. We add two equations to detect whether (or not) labels are well-bracketed in a context by neutralisation, (or nullification). For any labels $a \neq b$,

$$a \cdot \overline{a} = 1 \qquad a \cdot \overline{b} = 0. \tag{3}$$

Also, a is said *positive*, written $a > 0$, when it is empty or contains only positive labels.

A *levelled sharing context*, or simply an l-context, is a map γ from $\mathbb{N}$ to $\mathfrak{C}$, that is uniformly null: if for some $n \in \mathbb{N}$ we have $\gamma(n) = 0$, then $\gamma(m) = 0$ for any $m \in \mathbb{N}$. We write $(\gamma)|_n$ to denote the *restriction* of γ on n: if $m = n$ and $\gamma(m) \neq 0$, then $(\gamma)|_n(m) = 1$; otherwise $(\gamma)|_n(m) = \gamma(m)$. Also, we denote with $!^n a$ the *lifting* of a context a at level n. Namely, if $!^n a = \gamma$ then $\gamma(n) = a$, whilst $\gamma(m) = 1$ for any $m \neq n$. More precisely, $!^n a$ denotes $!(!^{n-1} a)$, where we set that $!1 = 1$, that $!0 = 0$, and also that $!(a \cdot b) = !a \cdot !b$. We say that γ is *positive*, written $\gamma > 0$, if $\gamma(n) > 0$ for any n. Given $\pi : u \rightsquigarrow v$ the *l-context* of π is defined as follows.

$$\mathfrak{c}\,(()) = 1 \tag{4}$$

$$\mathfrak{c}\,(\pi :: (u,v)) = \begin{cases} \mathfrak{c}\,(\pi) \cdot !^n a & \text{if there is } l = (u \mid\!+\!\rangle^n v) \text{ s.t. } \mathsf{CID}(l) = a \\ \mathfrak{c}\,(\pi) \cdot !^n \overline{a} & \text{if there is } l = (v \mid\!-\!\rangle^n u) \text{ s.t. } \mathsf{CID}(l) = a \\ (\mathfrak{c}\,(\pi))|_n & \text{if there is } (u \;?^n\; v) \\ \mathfrak{c}\,(\pi) & \text{if } u, v \text{ belong to a link of kind in } \{\overset{+}{\multimap}, \overset{-}{\multimap}, !\} \end{cases} \tag{5}$$

The *l-context of a vertex* in $U \in \mathsf{UG}$ is the context of any rooted path reaching it.

▶ **Proposition 28** (Positivity)**.** *Let π be a rooted downward path in $U \in \mathsf{UG}$. Then $\mathfrak{c}\,(\pi) > 0$.*

▶ **Proposition 29** (Path irrelevance)**.** *Let π, π' be two rooted paths in some UG ending with the vertex v. Then $\mathfrak{c}\,(\pi) = \mathfrak{c}\,(\pi')$.*

▶ **Definition 30** (Share and master)**.** A lift labelled with $x_{i:k}$ is *master* if $i = 0$, otherwise it is *shared*. A context a is master if a is empty or contains only master labels $x_{0:m}$; otherwise, it is a shared context. Finally, an l-context α is master if $\alpha(n)$ is master for any $n \in \mathbb{N}$, otherwise it is shared. A vertex is shared if its l-context is so, otherwise it is master; a link is shared if it has at least one shared vertex, otherwise it is master. A *share component* is a non-empty, connected, and maximal (w.r.t. inclusion) sub-graph whose vertices and links are shared. The set of the share components of $U \in \mathsf{UG}$ is denoted as $\mathsf{ShC}(U)$, their union is called the *share*, written $\mathsf{Sh}(U)$.

▶ **Definition 31** (Share boundary and interior)**.** Given $U \in \mathsf{UG}$, a shared lift $l = (u \mid\!*\!\rangle\ v)$ is a *boundary lift*, written $l \in \mathsf{bLft}(U)$, when $u \notin \mathsf{Sh}(U) \ni v$, whilst it is an *interior lift*, written $l \in \mathsf{iLft}(U)$, when $u, v \in \mathsf{Sh}(U)$. A given $v \in \mathsf{Sh}(U)$ is *boundary*, written $v \in \mathsf{BSh}(U)$, if it is the conclusion of a boundary lift, or it is linked by a lift to a boundary vertex. If additionally

Table 2 Metrics of UGR reduction: variation in the size of interior share, EPN-cost, variation in the number of boundary share components (Lemma 34); ASG-cost (Definition 35). Notations: μ is the reduction step, U is the net containing the redex, $d\kappa, r\kappa$ stands for a duplication rule; if involved: $h+1$ is the number of premisses of the ?-link, B is the box subnet, l, l' are the lifts. Also, intervals are enclosed in brackets, sets in braces.

Rule(s)	Proviso	$\Delta\mathsf{iSh}(\mu)$	$\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}(\mu)$	$\Delta\mathsf{BShC}(\mu)$	$\mathcal{C}^{\mathsf{ASG}}_{\mathsf{UG}}(\mu)$
$(\multimap)$	$\mu \notin \mathsf{Sh}(U)$	0	9	0	9
	$\mu \in \mathsf{Sh}(U)$	$-9 + \Delta\mathsf{BShC}(\mu)$	$18 - \Delta\mathsf{BShC}(\mu)$	$[0,2]$	0
$(!)$	$\mu \notin \mathsf{Sh}(U)$	0	6	0	6
	$\mu \in \mathsf{Sh}(U)$	-6	12	$[0,1]$	0
(tD)	$\mu \notin \mathsf{Sh}(U)$	$h \times \#\mathcal{E}(B) - h$	$3h+4$	$\{0,h\}$	$h+4$
	$\mu \in \mathsf{Sh}(U)$	$h \times \#\mathcal{E}(B) - 3h - 6$	$5h+10$	0	0
$(d!)$	$l \in \mathsf{bLft}(U)$	$-3 + \Delta\mathsf{BShC}(\mu)$	$3 - \Delta\mathsf{BShC}(\mu)$	$[0,1]$	3
$(d\multimap), (d\multimap), (r\multimap)$	$l \in \mathsf{bLft}(U)$	$-5 + \Delta\mathsf{BShC}(\mu)$	$5 - \Delta\mathsf{BShC}(\mu)$	$[0,2]$	5
$(d?), (r?)$	$l \in \mathsf{bLft}(U)$	$-2h - 3 + \Delta\mathsf{BShC}(\mu)$	$2h+3 - \Delta\mathsf{BShC}(\mu)$	$[0,h+1]$	$2h+3$
$(d\kappa), (r\kappa)$	$l \notin \mathsf{bLft}(U)$	0	0	0	0
(a)	$l, l' \in \mathsf{bLft}(U)$	0	0	-1	1
	otherwise	0	0	0	0
(s)	$l, l' \notin \mathsf{bLft}(U)$, $l, l' \in \mathsf{bLft}(\mu(U))$	$-1 + \Delta\mathsf{BShC}(\mu)$	$1 - \Delta\mathsf{BShC}(\mu)$	$[0,1]$	1
	otherwise	0	0	0	0
(m)	$l \in \mathsf{bLft}(U)$	0	0	-1	1
	otherwise	0	0	0	0

v is linked to a link other than a lift, then it is *boundary-limit*, written $v \in \mathsf{BLSh}(U)$. In such a case, the *boundary lift chain* of v is the longest sequence of lifts that induces a path from v to the conclusion of a boundary lift. If $\mathsf{Sh}(U) \ni v \notin \mathsf{BSh}(U)$ and v is a ι-vertex of a lift, then v is a *pseudo-boundary vertex*, the set of which is denoted by $\mathsf{bSh}(U)$. If $\mathsf{Sh}(U) \ni v \notin \mathsf{BSh}(U) \cup \mathsf{bSh}(U)$, then v is an *interior vertex*, the set of which is $\mathsf{iSh}(U)$. A share component having no interior vertices is a *boundary component*, and $\mathsf{BShC}(U)$ denotes the set of such components.

4.3.2 Reading proof-net cost and sharing cost on unshared reduction

Now we transpose $\mathcal{C}_{\mathsf{EPN}}$ and $\mathcal{C}_{\mathsf{ASG}}$ as two cost notions on UG reduction. *On the proof-net side* we define $\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}$, by subtracting from $\mathcal{C}_{\mathsf{EPN}}$ the variations in the size of the internal share. The first intuition is that the $\#\mathsf{iSh}(\cdot)$ represents a buffer for the amount of duplication work that is performed in big steps by EPN and delayed in small steps by ASG. In such a way, we obtain a definition of $\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}$ that is bounded by constant on any step, including (tD). The second intuition is that a reduction inside $\mathsf{iSh}(\cdot)$ cost up to twice the cost it would have $\mathcal{C}_{\mathsf{EPN}}$, since we need to account not only for the usual graph-rewriting cost, but also for the duplication cost previously performed of the involved links. Finally, we define and compute another notion of cost, $\mathcal{C}^{\mathsf{BShC}}_{\mathsf{UG}}$, which accounts the amount of reductions on boundary lifts in boundary share components. For the moment it only enables simplifications in the computation of $\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}$, but it will play a crucial role later.

▶ **Definition 32** (EPN metrics on UG)**.** Recall the notions of size and variation from Definition 16. Given $U \xrightarrow{\mu}_{\mathsf{UGR}} U'$, the *partial EPN-cost of* μ, is defined as $\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}(\mu) = \mathcal{C}_{\mathsf{EPN}}(\rho) - \Delta\mathsf{iSh}(\mu)$, while the *full* one is defined as $\overline{\mathcal{C}}^{\mathsf{EPN}}_{\mathsf{UG}}(\mu) = \mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}(\mu) + \#\mathsf{iSh}(U)$. The *boundary-share-components cost* of μ is $\mathcal{C}^{\mathsf{BShC}}_{\mathsf{UG}}(\mu) = |\Delta\mathsf{BShC}(\mu)|$.

▶ **Fact 33** (Correctness of $\overline{\mathcal{C}}^{\mathsf{EPN}}_{\mathsf{UG}}$)**.** *Let* $N \xrightarrow{\bar{\mu}}{}^{*}_{UG} U$ *and* $N \xrightarrow{\bar{\rho}}{}^{*}_{UG} N'$ *such that* $\bar{\mu} \mapsto \bar{\rho}$*. Then* $\mathcal{C}_{EPN}(\bar{\rho}) = \overline{\mathcal{C}}^{EPN}_{UG}(\bar{\mu})$.

▶ **Lemma 34** (Metrics on UGR-reduction). *Let μ be a UGR step. The possible values of $\Delta\mathsf{iSh}(\mu)$, $\mathcal{C}^{EPN}_{UG}(\mu)$ and $\Delta\mathsf{BShC}(\mu)$ are in Table 2.*

On the sharing graphs side we define $\mathcal{C}^{\mathsf{ASG}}_{\mathsf{UG}}$ as follows: logical redexes are accounted for only if they are a master copy; other redexes are accounted for only if the involved lift is a boundary one. Intuitively, we distribute a $k+1$-ary mux duplication into k propagations of boundary lifts, and a step of another kind into the unique *master copy* of its redex.

▶ **Definition 35** (Cost on unshared graph). Let μ be a UGR-reduction step. The *ASG-cost* of μ, written $\mathcal{C}^{\mathsf{ASG}}_{\mathsf{UG}}(\mu_i)$, is defined in the rightmost column of Table 2.

Let $N \in EPN$, $U \in UG$ and $G \in ASG$ such that $N \xrightarrow{\bar{\sigma}}{}^{*}_{ASG} G$ and $N \xrightarrow{\bar{\mu}}{}^{*}_{UGR} U$, with $\bar{\sigma} \hookrightarrow \bar{\mu}$.

▶ **Lemma 36** (Master copy). *If $v \in V(G)$ and $V' \leftarrow v$, then V' contains a unique master.*

▶ **Lemma 37** (Correctness of $\mathcal{C}^{\mathsf{ASG}}_{\mathsf{UG}}$). $\mathcal{C}_{ASG}(\bar{\sigma}) = \mathcal{C}^{ASG}_{UG}(\bar{\mu})$.

4.3.3 Comparison of the two unshared costs

Finally, we now compare $\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}(\bar{\mu})$ and $\mathcal{C}^{\mathsf{ASG}}_{\mathsf{UG}}(\bar{\mu})$, and find their difference to be bounded by $\mathcal{C}^{\mathsf{BShC}}_{\mathsf{UG}}(\bar{\mu})$. By a mere local observation we find a linear bound in $\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}$ for the portion of $\mathcal{C}^{\mathsf{BShC}}_{\mathsf{UG}}$ induced by logical, duplicating, merging and annihilation rules. For the portion caused by swap rules, instead, we find a quadratic limitation. Therefore, the overhead of ASG with respect to its EPN simulation admits a quadratic bound.

▶ **Lemma 38.** *Let $N \in EPN$, $U \in UG$, s.t. $N \xrightarrow{\bar{\mu}}{}^{*}_{UGR} U$. Then $\mathcal{C}^{ASG}_{UG}(\bar{\mu}) - \mathcal{C}^{\mathsf{BShC}}_{UG}(\bar{\mu}) \leq \mathcal{C}^{EPN}_{UG}(\bar{\mu})$.*

▶ **Lemma 39** (Bound to $\mathcal{C}^{\mathsf{BShC}}_{\mathsf{UG}}$). *For any $N \in EPN$ and any sequence $\bar{\mu}$ of UGR-reduction on N, there exists a quadratic function q such that $\mathcal{C}^{\mathsf{BShC}}_{UG}(\bar{\mu}) \leq q(\mathcal{C}^{EPN}_{UG}(\bar{\mu}))$.*

▶ **Theorem 40** (Complexity comparison). *Let $N, N' \in EPN$, $G \in ASG$ such that $N \xrightarrow{\bar{\sigma}}{}^{*}_{ASGR} G$ and $N \xrightarrow{\bar{\rho}}{}^{*}_{EPN} N'$, where $\bar{\rho} \twoheadrightarrow \bar{\sigma}$. Then $\mathcal{C}_{ASG}(\bar{\sigma}) \leq q\,(\mathcal{C}_{EPN}(\bar{\rho}))$ where q is a quadratic function.*

5 Conclusions

Two reflections and four questions emerge from the study we have presented.

Discussion

1. *A quite positive partial answer to the efficiency of sharing graphs* comes from the quadratic upper-limit to the complexity of their reductions. This is motivated by two arguments.
   - **a.** Hypotheses of our complexity measurement were purposely extremely conservative. Indeed, we not only consider the read-back – an unavoidable portion of the reduction work – by directly including their rewriting rules. But we also allow these rules to be applied freely, also before the end of the β-normalisation, thus allowing duplications of redexes in a sub-optimal fashion [20].
   - **b.** The worst-case overhead of sharing graphs are usually counterbalanced by other benefits. Laziness in the strategy of duplication, for instance, has shown speed-ups up to exponential size [3]. Locality and asynchronicity of the computational model, moreover, allow parallelisable implementation with little effort.

2. *The cost of local duplication is legitimate.* Normalisation with sharing graphs of some ELL-typed λ-terms may cause an elementary explosion in the number of local duplication rules (mux propagations) [2]. This should not surprise, because simply-typed terms in general may require an implementation cost that is more than elementary [22]. To further clarify this point, Lemma 38 shows that duplications performed by sharing graphs have a cost that is linearly bounded by the cost of proof-net reduction.

Open questions

1. Can we improve the quadratic bound? Or there is instead a λ-term typeable in ELL or LLL whose sharing normalisation requires indeed a cost that is quadratic with respect to proof-nets normalisation?
2. Does a similar complexity upper-bound hold for the more general cases of λ-calculus and MELL?
3. Complementarily, is there also a lower bound giving theoretical evidence of performance gains?
4. Can our bound be directly related to the number of β-steps in the leftmost-outermost strategy on the λ-calculus [9]? Are sharing graphs themselves are a reasonable and parsimonious cost model the λ-calculus?

References

1 Andrea Asperti. About the efficient reduction of lambda terms. *arXiv*, January 2017. 1701.04240. URL: `http://arxiv.org/abs/1701.04240`.

2 Andrea Asperti, Paolo Coppola, and Simone Martini. (Optimal) duplication is not elementary recursive. *Inform. and Comput.*, 193(1):21–56, 2004. `doi:10.1016/j.ic.2004.05.001`.

3 Andrea Asperti, Cecilia Giovannetti, and Andrea Naletto. The Bologna optimal higher-order machine. *Journal of Functional Programming*, 6(6):763–810, November 1996. `https://github.com/asperti/BOHM1.1`. `doi:10.1017/S0956796800001994`.

4 Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*, volume 45 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

5 Andrea Asperti and Harry G. Mairson. Parallel Beta Reduction Is Not Elementary Recursive. *Inf. Comput.*, 170(1):49–80, 2001. `doi:10.1006/inco.2001.2869`.

6 Andrea Asperti and Luca Roversi. Intuitionistic light affine logic. *ACM Trans. Comput. Logic*, 3:137–175, January 2002. `doi:10.1145/504077.504081`.

7 Patrick Baillot, Paolo Coppola, and Ugo Dal Lago. Light logics and optimal reduction: Completeness and complexity. *Information and Computation*, 209(2):118–142, February 2011. `doi:10.1016/j.ic.2010.10.002`.

8 Ugo Dal Lago. Context semantics, linear logic, and computational complexity. *ACM Transactions on Computational Logic (TOCL)*, 10(4):25:1–25:32, August 2009. `doi:10.1145/1555746.1555749`.

9 Ugo Dal Lago and Beniamino Accattoli. (Leftmost-Outermost) Beta Reduction is Invariant, Indeed. *Logical Methods in Computer Science*, 12, 2016. `doi:10.2168/LMCS-12(1:4)2016`.

10 Ugo Dal Lago and Simone Martini. On Constructor Rewrite Systems and the Lambda-Calculus. In *Automata, Languages and Programming*, pages 163–174. Springer, July 2009. `doi:10.1007/978-3-642-02930-1_14`.

11 J.-Y. Girard. Geometry of interaction I. Interpretation of system F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Logic colloquium 1988*, volume 127 of *Studies in*

Logic and The Foundations of Mathematics, pages 221–260. North-Holland, 1989. doi:10.2277/978-0521621120.

12 Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998. doi:10.1006/inco.1998.2700.

13 Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The geometry of optimal lambda reduction. In *Conference record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'92)*, pages 15–26, Albequerque, New Mexico, January 1992.

14 Stefano Guerrini, Thomas Leventis, and Marco Solieri. Deep into optimality – complexity and correctness of sharing implementation of bounded logics. Third International Workshop on Developments in Implicit Complexity, 2012.

15 Stefano Guerrini, Simone Martini, and Andrea Masini. Modal Logic, Linear Logic, Optimal Lambda-Reduction. In *Logic and Foundations of Mathematics*, number 280 in Synthese Library, pages 271–282. Springer, 1999. doi:10.1007/978-94-017-2109-7_20.

16 Stefano Guerrini, Simone Martini, and Andrea Masini. Coherence for sharing proof-nets. *Theoretical Computer Science*, 294(3):379–409, February 2003. doi:10.1016/S0304-3975(01)00162-1.

17 Pavol Hell and Jaroslav Nesetril. *Graphs and homomorphisms*. Oxford Univ. Press, 2004.

18 Yves Lafont. Interaction nets. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL'90, pages 95–108, New York, NY, USA, 1990. ACM. doi:10.1145/96709.96718.

19 John Lamping. An algorithm for optimal lambda calculus reduction. In *Proc. of Seventeenth Annual ACM Symposyum on Principles of Programming Languages*, pages 16–30, San Francisco, California, January 1990. doi:10.1145/96709.96711.

20 Jean-Jacques Lévy. Optimal reductions in the lambda-calculus. In Jonathan P. Seldin and J. Roger Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 159–191. Academic Press, 1980.

21 Simon L. Peyton Jones. Implementing lazy functional languages on stock hardware: the Spineless Tagless G-machine. *Journal of Functional Programming*, 2(2):127–202, April 1992. doi:10.1017/S0956796800000319.

22 Richard Statman. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9(1):73–81, 1979. doi:10.1016/0304-3975(79)90007-0.

23 Peter van Emde Boas. *Machine models and simulations, Handbook of theoretical computer science (vol. A): algorithms and complexity*. MIT Press, Cambridge, MA, 1991.

24 C. P. Wadsworth. *Semantics and pragmatics of the lambda-calculus*. PhD thesis, University of Oxford, 1971.

Continuation Passing Style for Effect Handlers*

Daniel Hillerström[1], Sam Lindley[2], Robert Atkey[3], and K. C. Sivaramakrishnan[4]

1 **The University of Edinburgh, Edinburgh, UK**
daniel.hillerstrom@ed.ac.uk
2 **The University of Edinburgh, Edinburgh, UK**
sam.lindley@ed.ac.uk
3 **University of Strathclyde, Strathclyde, UK**
robert.atkey@strath.ac.uk
4 **University of Cambridge, Cambridge, UK**
sk826@cl.cam.ac.uk

Abstract

We present Continuation Passing Style (CPS) translations for Plotkin and Pretnar's effect handlers with Hillerström and Lindley's row-typed fine-grain call-by-value calculus of effect handlers as the source language. CPS translations of handlers are interesting theoretically, to explain the semantics of handlers, and also offer a practical implementation technique that does not require special support in the target language's runtime.

We begin with a first-order CPS translation into untyped lambda calculus which manages a stack of continuations and handlers as a curried sequence of arguments. We then refine the initial CPS translation first by uncurrying it to yield a properly tail-recursive translation and second by making it higher-order in order to contract administrative redexes at translation time. We prove that the higher-order CPS translation simulates effect handler reduction. We have implemented the higher-order CPS translation as a JavaScript backend for the Links programming language.

1998 ACM Subject Classification D.3.3 Language Constructs and Features, F.3.2 Semantics of Programming Languages

Keywords and phrases effect handlers, delimited control, continuation passing style

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.18

1 Introduction

Algebraic effects, introduced by Plotkin and Power [25], and their handlers, introduced by Plotkin and Pretnar [26], provide a modular foundation for effectful programming. Though originally studied in a theoretical setting, effect handlers are also of practical interest, as witnessed by a range of recent implementations [2, 4, 8, 11, 13, 15, 17, 20]. Notably, the Multicore OCaml project brings effect handlers to the OCaml programming language as a means for abstracting over different scheduling strategies [8]. As a programming abstraction, effect handlers can be viewed as a more modular alternative to monads [23, 29].

An algebraic effect is a signature of operations along with an equational theory on those operations. An effect handler is a delimited control operator which *interprets* a particular

* Daniel Hillerström was supported by EPSRC grant EP/L01503X/1 (CDT in Pervasive Parallelism). Sam Lindley was supported by EPSRC grant EP/K034413/1 (A Basis for Concurrency and Distribution). K. C. Sivaramakrishnan was supported by a Research Fellowship from the Royal Commission for the Exhibition of 1851.

subset of the signature of operations up to equivalences demanded by the equational theory. In practice current implementations do not support equations on operations, and as such, the underlying algebra is the free algebra, allowing handlers maximal interpretative freedom. Correspondingly, in this paper we assume free algebras for effects.

There are many feasible implementation strategies for effect handlers. For instance, Kammar et al.'s libraries make use variously of free monads, continuation monads, and delimited continuations [13]; the server backend of the Links programming language uses an abstract machine [11]; and Multicore OCaml relies on explicit manipulation of the runtime stack [8]. Explicit stack manipulation is appealing when one has complete control over the design of the backend. Similarly, delimited continuations are appealing when the backend already has support for delimited continuations [13, 16]. However, if the backend does not support delimited continuations or explicit stack manipulation, for instance when targeting a backend language like JavaScript, an alternative approach is necessary.

In this paper we study how to translate effect handlers from a rich source lambda calculus into a plain lambda calculus without necessarily requiring additional primitives. Specifically, we study *continuation passing style* (CPS) translations for handlers. CPS does not extend the lambda calculus, rather, CPS amounts to using a restricted subset of the plain lambda calculus. Furthermore CPS is an established intermediate representation used by compilers [1, 14], making it a realistic compilation target, and it provides a general framework for implementing control flow, making it a good fit for implementing control operators such as effect handlers. The only existing CPS translation for effect handlers we are aware of is due to Leijen [17]. His work differs from ours in that he does not CPS translate away operations or handlers, but rather uses a CPS translation to lift code into a free monad, relying on a special handle primitive in the runtime. Leijen's formalism includes features that we do not. In particular, he performs a selective CPS translation in order to avoid overhead in code that does not use algebraic effects. We have implemented a JavaScript backend for Links, based on our formalism, which also performs a selective CPS translation. In this paper we do not formalise the selective aspect of the translation as it is mostly orthogonal.

This paper makes the following contributions:

1. A concise curried CPS translation for effect handlers (§4.2.1). The translation has two shortcomings: it is not properly tail-recursive and it yields administrative redexes.
2. A higher-order uncurried variant of the curried CPS translation, which is properly tail-recursive and partially evaluates administrative redexes at translation time (§4.3).
3. A correctness proof for the higher-order CPS translation (§4.3).
4. An implementation of the higher-order CPS translation as a backend for Links [5] (§5).

The paper proceeds as follows. §2 gives a primer to programming with effect handlers. §3 describes a core calculus of algebraic effects and handlers. §4 presents the CPS translations, correctness proof, and variations. §5 briefly describes our implementation. §6 concludes.

2 Modular interpretation of effectful computations

In this section we give a flavour of programming with algebraic effects and handlers. We use essentially the syntax of our core calculus (§3) along with syntactic sugar to make the examples more readable (in particular, we use direct-style rather than fine-grain call-by-value). We consider two effects: *nondeterminism* and *exceptions*, the former given by a nondeterministic choice operation Choose; the latter by an exception-raising operation Fail.

We combine nondeterminism and exceptions to model a drunk attempting to toss a coin. The coin toss and whether it succeeds is modelled by the Choose operation. The drunk

dropping the coin is modelled by the Fail operation.

```
drunkToss : Toss ! {Choose : Bool; Fail : Zero}
drunkToss = if do Choose then
                if do Choose then Heads else Tails
            else absurd do Fail
```

This code declares an *abstract computation* drunkToss, which potentially invokes two abstract operations Choose and Fail using the **do** primitive. The type signature of drunkToss reads: drunkToss is a computation with effect signature {Choose : Bool; Fail : Zero} and return value Toss, whose constructors are Heads and Tails. The order of operation names in the effect signature is irrelevant. The first invocation of Choose decides whether the coin is caught, while the second invocation decides the face. The Fail operation never returns, so its return type is Zero, which is eliminated by the **absurd** construct.

A possible interpretation of drunkToss uses lists to model nondeterminism, where the return operation lifts its argument into a singleton list, the choose handler concatenates lists of possible outcomes, and the fail operation returns the empty list:

```
nondet : α ! {Choose : Bool; Fail : Zero} ⇒ List α ! {}
nondet = return x ↦ [x]
         Choose r ↦ r True ++ r False
         Fail r ↦ []
```

The type signature conveys that the handler transforms an abstract computation into a concrete computation where the operations Choose and Fail are instantiated. The handler comprises three clauses. The return clause specifies how to handle the return value of the computation. The other two clauses specify how to interpret the operations. The Choose clause binds a *resumption* r, a function which captures the delimited continuation of the operation Choose. The interpretation of Choose explores both alternatives by invoking the resumption twice and concatenating the results. The Fail clause ignores the provided resumption and returns simply the empty list, []. Thus handling drunkToss with nondet yields all possible positive outcomes, i.e. [Heads, Tails].

A key feature of effect handlers is that the use of an operation is *decoupled* from its interpretation. Rather than having one handler which handles every operation, we may opt for a more fine-grained approach using multiple handlers which each instantiate a subset of the abstract operations. For example, we can define handlers for each of the two operations.

```
allChoices : α ! {Choose : Bool; ρ} ⇒ List α ! {Choose : θ; ρ}
allChoices = return x ↦ [x]
             Choose r ↦ r True ++ r False
```

This effect signature differs from the signature of nondet; it mentions only one operation and it mentions an *effect variable* variable ρ which ranges over all unmentioned operations. In addition Choose is mentioned in output effect signature. (This is because we adopt a Remy-style row-type system [28] in which negative information is made explicit. At the cost of slightly less expressivity, it is possible to eliminate these effects in the output type by permitting effect shadowing as in Koka [17] and Frank [20].) The notation Choose : θ denotes that the operation may or may not appear again. This handler implicitly *forwards* Fail to another enclosing handler such as the following.

```
failure : α ! {Fail : Zero; ρ} ⇒ List α ! {Fail : θ; ρ}
failure = return x ↦ [x]
          Fail r ↦ []
```

Value types	$A, B ::= A \to C \mid \forall \alpha^K.C$	Types	$T ::= A \mid C \mid E \mid R \mid P \mid F$
	$\mid \langle R \rangle \mid [R] \mid \alpha$	Kinds	$K ::= \mathsf{Type} \mid \mathsf{Row}_{\mathcal{L}} \mid \mathsf{Presence}$
Computation types	$C, D ::= A!E$		$\mid \mathsf{Comp} \mid \mathsf{Effect} \mid \mathsf{Handler}$
Effect types	$E ::= \{R\}$	Label sets	$\mathcal{L} ::= \emptyset \mid \{\ell\} \uplus \mathcal{L}$
Row types	$R ::= \ell : P; R \mid \rho \mid \cdot$	Type environments	$\Gamma ::= \cdot \mid \Gamma, x : A$
Presence types	$P ::= \mathsf{Pre}(A) \mid \mathsf{Abs} \mid \theta$	Kind environments	$\Delta ::= \cdot \mid \Delta, \alpha : K$
Handler types	$F ::= C \Rightarrow D$		

Figure 1 Types, effects, kinds, and environments.

Now, we have two possible compositions, and we must tread carefully as they are semantically different. For example, **handle** (**handle** drunkToss **with** allChoices) **with** failure yields the empty list. But **handle** (**handle** drunkToss **with** failure) **with** allChoices yields all possible outcomes, i.e. [[Heads], [Tails], []] where the empty list conveys failure. The behaviour of nondet can be obtained by concatenating the result of the latter composition. Effect handlers also permit multiple interpretations of the same abstract computation.

3 A calculus of handlers and rows

In this section, we recapitulate our Church-style row-polymorphic call-by-value calculus for effect handlers $\lambda^{\rho}_{\text{eff}}$ (pronounced "lambda-eff-row") [11].

The design of $\lambda^{\rho}_{\text{eff}}$ is inspired by the λ-calculi of Kammar et al. [13], Pretnar [27], and Lindley and Cheney [19]. As in the work of Kammar et al. [13], each handler can have its own effect signature. As in the work of Pretnar [27], the underlying formalism is fine-grain call-by-value [18], which names each intermediate computation like in A-normal form [9], but unlike A-normal form is closed under β-reduction. As in the work of Lindley and Cheney [19], the effect system is based on row polymorphism.

3.1 Types

The syntax of types, kinds, label sets, and type and kind environments is given in Figure 1.

Value types. The function type $A \to C$ maps values of type A to computations of type C. The polymorphic type $\forall \alpha^K.\, C$ is parameterised by a type variable α of kind K. The record type $\langle R \rangle$ represents records with fields constrained by row R. Dually, the variant type $[R]$ represents tagged sums constrained by row R.

Computation types. The computation type $A!E$ is given by a value type A and an effect type E, which specifies the operations a computation inhabiting this type may perform.

Row types. Effect, record, and variant types are defined in terms of rows. A row type embodies a collection of distinct labels, each of which is annotated with a presence type. A presence type indicates whether a label is *present* with some type A ($\mathsf{Pre}(A)$), *absent* (Abs) or *polymorphic* in its presence (θ). Row types are either *closed* or *open*. A closed row type ends in $\cdot$, whilst an open row type ends with a *row variable* ρ. The row variable in an open row can be instantiated with additional labels. We identify rows up to reordering of labels. For instance, we consider the rows $\ell_1 : P_1; \cdots; \ell_n : P_n; \cdot$ and $\ell_n : P_n; \cdots; \ell_1 : P_1; \cdot$ equivalent. Absent labels in closed rows are redundant: if R is closed, then $\ell : \mathsf{Abs}; R$ is equivalent to R. The unit and empty type are definable in terms of row types. We define the unit type as the

$$
\begin{array}{lll}
\text{Values} & V, W & ::= x \mid \lambda x^{A}.\, M \mid \Lambda \alpha^{K}.\, M \mid \langle\rangle \mid \langle \ell = V; W \rangle \mid (\ell\ V)^{R} \\
\text{Computations} & M, N & ::= V\ W \mid V\ T \\
& & \mid \mathbf{let}\ \langle \ell = x; y \rangle = V\ \mathbf{in}\ N \mid \mathbf{case}\ V \{ \ell\ x \mapsto M; y \mapsto N \} \mid \mathbf{absurd}^{C}\ V \\
& & \mid \mathbf{return}\ V \mid \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N \\
& & \mid (\mathbf{do}\ \ell\ V)^{E} \mid \mathbf{handle}\ M\ \mathbf{with}\ H \\
\text{Handlers} & H & ::= \{\mathbf{return}\ x \mapsto M\} \mid \{\ell\ p\ r \mapsto M\} \uplus H
\end{array}
$$

Figure 2 Term syntax.

empty, closed record, that is, $\langle\cdot\rangle$. Similarly, we define the empty type as the empty, closed variant $[\cdot]$. Often we omit the $\cdot$ for closed rows.

Handler types. The handler type $C \Rightarrow D$ represents handlers that transform computations of type C into computations of type D.

Kinds. We have six kinds: Type, Comp, Effect, $\mathsf{Row}_{\mathcal{L}}$, Presence, Handler, which respectively classify value types, computation types, effect types, row types, presence types, and handler types. Row kinds are annotated with a set of labels $\mathcal{L}$. The kind of a complete row is $Row_{\emptyset}$. More generally, the kind $Row_{\mathcal{L}}$ denotes a partial row that cannot mention the labels in $\mathcal{L}$. We write $\ell : A$ as syntactic sugar for $\ell : \mathsf{Pre}(A)$.

Type variables. We let α, ρ and θ range over type variables. By convention we use α for value type variables or for type variables of unspecified kind, ρ for type variables of row kind, and θ for type variables of presence kind.

Type and kind environments. Type environments (Γ) map term variables to their types and kind environments (Δ) map type variables to their kinds.

3.2 Terms

The terms are given in Figure 2. We let x, y, z, r, p range over term variables. By convention, we use r to denote resumption names. The syntax partitions terms into values, computations and handlers. Value terms comprise variables (x), lambda abstraction ($\lambda x^{A}.\, M$), type abstraction ($\Lambda \alpha^{K}.\, M$), and the introduction forms for records and variants. Records are introduced using the empty record $\langle\rangle$ and record extension $\langle \ell = V; W \rangle$, whilst variants are introduced using injection $(\ell\ V)^{R}$, which injects a field with label ℓ and value V into a row whose type is R. The annotation supports bottom-up type reconstruction.

All elimination forms are computation terms. Abstraction and type abstraction are eliminated using application ($V\ W$) and type application ($V\ T$) respectively. The record eliminator ($\mathbf{let}\ \langle \ell = x; y \rangle = V\ \mathbf{in}\ N$) splits a record V into x, the value associated with ℓ, and y, the rest of the record. Non-empty variants are eliminated using the case construct ($\mathbf{case}\ V\ \{\ell\ x \mapsto M; y \mapsto N\}$), which evaluates the computation M if the tag of V matches ℓ. Otherwise it falls through to y and evaluates N. The elimination form for empty variants is ($\mathbf{absurd}^{C}\ V$). A trivial computation ($\mathbf{return}\ V$) returns value V. The expression ($\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N$) evaluates M and binds the result value to x in N.

The construct $(\mathbf{do}\ \ell\ V)^{E}$ invokes an operation ℓ with value argument V. The handle construct ($\mathbf{handle}\ M\ \mathbf{with}\ H$) runs a computation M with handler definition H. A handler definition H consists of a return clause $\{\mathbf{return}\ x \mapsto M\}$ and a possibly empty set of operation clauses $\{\ell\ p\ r \mapsto N_{\ell}\}_{\ell \in \mathcal{L}}$. The return clause defines how to handle the final return

value of the handled computation, which is bound to x in M. The operation clause for ℓ binds the operation parameter to p and the resumption r in N_ℓ.

We define three projections on handlers: H^{ret} yields the singleton set containing the return clause of H and H^ℓ yields the set of either zero or one operation clauses in H that handle the operation ℓ and H^{ops} yields the set of all operation clauses in H. We write $dom(H)$ for the set of operations handled by H. As our calculus is Church-style, we annotate various term forms with type or kind information; (term abstraction, type abstraction, injection, operations, and empty cases); we sometimes omit these annotations.

Syntactic sugar. We make use of standard syntactic sugar for pattern matching, n-ary record extension, n-ary case elimination, and n-ary tuples. For instance:

$$\begin{aligned}
\lambda\langle x, y\rangle.M &\equiv \lambda z.\mathbf{let}\ \langle x, y\rangle = z\ \mathbf{in}\ M \\
\langle V_1, \ldots, V_n\rangle &\equiv \langle 1 = V_1, \ldots, n = V_n\rangle \\
\mathbf{case}\ V\ \{\ell_1\ x \mapsto N_1; &\equiv \mathbf{case}\ V\ \{\ell_1\ x \mapsto N_1; z \mapsto \mathbf{case}\ z\ \{\ell_2\ x \mapsto N_1; z \mapsto \\
\quad \ldots; & \qquad \ldots \\
\quad \ell_n\ x \mapsto N_n; z \mapsto N\} & \qquad \mathbf{case}\ z\ \{\ell_n\ x \mapsto N_1; z \mapsto N\} \ldots\}
\end{aligned}$$

3.3 Kinding and typing

The kinding judgement $\Delta \vdash T : K$ states that type T has kind K in kind environment Δ. The value typing judgement $\Delta; \Gamma \vdash V : A$ states that value term V has type A under kind environment Δ and type environment Γ. The computation typing judgement $\Delta; \Gamma \vdash M : C$ states that term M has computation type C under kind environment Δ and type environment Γ. The handler typing judgement $\Delta; \Gamma \vdash H : C \Rightarrow D$ states that handler H has type $C \Rightarrow D$ under kind environment Δ and type environment Γ. In the typing judgements, we implicitly assume that Γ, A, C, and D, are well-kinded with respect to Δ. We define the function $FTV(\Gamma)$ to be the set of free type variables in Γ. The full kinding and typing rules are given in Appendix A. The interesting rules are T-Do, T-Handle, and T-Handler.

T-Do
$$\frac{\Delta; \Gamma \vdash V : A \qquad E = \{\ell : A \to B; R\}}{\Delta; \Gamma \vdash (\mathbf{do}\ \ell\ V)^E : B!E}$$

T-Handle
$$\frac{\Delta; \Gamma \vdash M : C \qquad \Delta; \Gamma \vdash H : C \Rightarrow D}{\Delta; \Gamma \vdash \mathbf{handle}\ M\ \mathbf{with}\ H : D}$$

T-Handler
$$\frac{\begin{array}{c} C = A!\{(\ell_i : A_i \to B_i)_i; R\} \qquad D = B!\{(\ell_i : P_i)_i; R\} \qquad H = \{\mathbf{return}\ x \mapsto M\} \uplus \{\ell_i\ p\ r \mapsto N_{\ell_i}\}_i \\ \Delta; \Gamma, x : A \vdash M : D \qquad [\Delta; \Gamma, p : A_i, r : B_i \to D \vdash N_{\ell_i} : D]_i \end{array}}{\Delta; \Gamma \vdash H : C \Rightarrow D}$$

The T-Handler rule is where most of the work happens. The effect rows on the computation type C and the output computation type D must share the same suffix R. This means that the effect row of D must explicitly mention each of the operations ℓ_i to say whether an ℓ_i is present with a given type signature, absent, or polymorphic in its presence. The row R describes the operations that are forwarded. It may include a row-variable, in which case an arbitrary number of effects may be forwarded by the handler.

3.4 Operational semantics

We give a small-step operational semantics for $\lambda^\rho_{\text{eff}}$. Figure 3 gives the reduction rules. The reduction relation $\rightsquigarrow$ is defined on computation terms. We use evaluation contexts ($\mathcal{E}$) to focus on the active expression. The interesting rules are the handler rules. We write $BL(\mathcal{E})$ for the set of operation labels bound by $\mathcal{E}$.

$$BL([\,]) = \emptyset \qquad BL(\mathbf{let}\ x \leftarrow \mathcal{E}\ \mathbf{in}\ N) = BL(\mathcal{E}) \qquad BL(\mathbf{handle}\ \mathcal{E}\ \mathbf{with}\ H) = BL(\mathcal{E}) \cup dom(H)$$

$$
\begin{array}{lrll}
\text{S-App} & (\lambda x^A.\, M)\, V & \rightsquigarrow & M[V/x] \\
\text{S-TyApp} & (\Lambda \alpha^K.\, M) A & \rightsquigarrow & M[A/\alpha] \\
\text{S-Split} & \mathbf{let}\ \langle \ell = x; y \rangle = \langle \ell = V; W \rangle\ \mathbf{in}\ N & \rightsquigarrow & N[V/x, W/y] \\
\text{S-Case}_1 & \mathbf{case}\ (\ell\ V)^R\{\ell\ x \mapsto M; y \mapsto N\} & \rightsquigarrow & M[V/x] \\
\text{S-Case}_2 & \mathbf{case}\ (\ell\ V)^R\{\ell'\ x \mapsto M; y \mapsto N\} & \rightsquigarrow & N[(\ell\ V)^R/y], \quad \text{if } \ell \neq \ell' \\
\text{S-Let} & \mathbf{let}\ x \leftarrow \mathbf{return}\ V\ \mathbf{in}\ N & \rightsquigarrow & N[V/x] \\
\text{S-Handle-Ret} & \mathbf{handle}\ (\mathbf{return}\ V)\ \mathbf{with}\ H & \rightsquigarrow & N[V/x], \quad \text{where } H^{\text{ret}} = \{\mathbf{return}\ x \mapsto N\} \\
\text{S-Handle-Op} & \mathbf{handle}\ \mathcal{E}[\mathbf{do}\ \ell\ V]\ \mathbf{with}\ H & \rightsquigarrow & N[V/p, \lambda y.\, \mathbf{handle}\ \mathcal{E}[\mathbf{return}\ y]\ \mathbf{with}\ H/r], \\
 & & & \quad \text{where } \ell \notin BL(\mathcal{E}) \text{ and } H^\ell = \{\ell\ p\ r \mapsto N\} \\
\text{S-Lift} & \mathcal{E}[M] & \rightsquigarrow & \mathcal{E}[N], \quad \text{if } M \rightsquigarrow N
\end{array}
$$

Evaluation contexts $\mathcal{E} ::= [\,] \mid \mathbf{let}\ x \leftarrow \mathcal{E}\ \mathbf{in}\ N \mid \mathbf{handle}\ \mathcal{E}\ \mathbf{with}\ H$

Figure 3 Small-step operational semantics.

The rule S-Handle-Ret invokes the return clause of a handler. The rule S-Handle-op handles an operation by invoking the appropriate operation clause. The constraint $\ell \notin BL(\mathcal{E})$ ensures that no inner handler inside the evaluation context is able to handle the operation: thus a handler is able to reach past any other inner handlers that do not handle ℓ.

We write R^+ for the transitive closure of relation R.

▶ **Definition 1.** We say that computation term N is normal with respect to effect E, if N is either of the form $\mathbf{return}\ V$, or $\mathcal{E}[\mathbf{do}\ \ell\ W]$, where $\ell \in E$ and $\ell \notin BL(\mathcal{E})$.

▶ **Theorem 2** (Type Soundness). *If $\vdash M : A!E$, then there exists $\vdash N : A!E$, such that $M \rightsquigarrow^+ N \not\rightsquigarrow$, and N is normal with respect to effect E.*

4 CPS translations

We now turn to the main business of the paper: continuation passing style translations from a calculus with effects and handlers to a calculus with neither. In doing so, we achieve two aims. First, we give an alternative formal explanation of effect handlers' semantics independent of the standard free monad interpretation. Second, we offer an implementation technique that is more efficient than the free monad interpretation because it does not allocate intermediate computation trees. We present our CPS translation in stages. We start with a basic translation for fine-grain call-by-value without handlers in §4.1. We then formulate first-order translations that progressively move from representing the dynamic stack of handlers as functions to explicit stacks in §4.2. This prepares us for our final higher-order one-pass translation in §4.3 that uses static computation at translation time to avoid administrative reductions during runtime. We then consider shallow handlers in §4.4, and exceptions in §4.5.

The untyped target calculus for our CPS translations is given in Figure 4. As in our fine-grain call-by-value source language, we make a syntactic distinction between values and computations. The reductions are standard β-reductions, also given in Figure 4. There are three differences from fine-grain call-by-value: *i)* we have no explicit **return** to lift values to computations, value terms are silently included in computation terms; *ii)* there is no **let** in the target calculus, because all sequencing will be expressed via continuation passing; and *iii)* we permit the function position of an application to be a computation (i.e., the application form is $M\ W$ rather than $V\ W$). This latter relaxation is used in our initial CPS translations, but will be ruled out in our final translation.

Syntax

$$
\begin{array}{lll}
\text{Values} & V, W & ::= x \mid \lambda x.M \mid \langle\rangle \mid \langle \ell = V; W\rangle \mid \ell\ V \\
\text{Computations} & M, N & ::= V \mid M\ W \mid \mathbf{let}\ \langle \ell = x; y\rangle = V\ \mathbf{in}\ N \\
& & \quad \mid\ \mathbf{case}\ V\,\{\ell\ x \mapsto M; y \mapsto N\} \mid \mathbf{absurd}\ V \\
\text{Evaluation contexts} & \mathcal{E} & ::= [\,] \mid \mathcal{E}\ W
\end{array}
$$

Reductions

$$
\begin{array}{lr}
\text{U-App} & (\lambda x.\, M)\, V \rightsquigarrow M[V/x] \\
\text{U-Split} & \mathbf{let}\ \langle \ell = x; y\rangle = \langle \ell = V; W\rangle\ \mathbf{in}\ N \rightsquigarrow N[V/x, W/y] \\
\text{U-Case}_1 & \mathbf{case}\ (\ell\ V)\{\ell\ x \mapsto M; y \mapsto N\} \rightsquigarrow M[V/x] \\
\text{U-Case}_2 & \mathbf{case}\ (\ell\ V)\{\ell'\ x \mapsto M; y \mapsto N\} \rightsquigarrow N[\ell\ V/y], \quad \text{if } \ell \neq \ell' \\
\text{U-Lift} & \mathcal{E}[M] \rightsquigarrow \mathcal{E}[N], \quad \text{if } M \rightsquigarrow N
\end{array}
$$

Figure 4 Untyped target calculus.

Values

$$
\begin{array}{rl}
[\![x]\!] &= x \\
[\![\lambda x.M]\!] &= \lambda x.[\![M]\!] \\
[\![\Lambda\alpha.M]\!] &= \lambda k.[\![M]\!]\ k \\
[\![\langle\rangle]\!] &= \langle\rangle \\
[\![\langle \ell = V; W\rangle]\!] &= \langle \ell = [\![V]\!]; [\![W]\!]\rangle \\
[\![\ell\ V]\!] &= \ell\ [\![V]\!]
\end{array}
$$

Computations

$$
\begin{array}{rl}
[\![V\ W]\!] &= [\![V]\!]\ [\![W]\!] \\
[\![V\ A]\!] &= [\![V]\!] \\
[\![\mathbf{let}\ \langle \ell = x; y\rangle = V\ \mathbf{in}\ N]\!] &= \mathbf{let}\ \langle \ell = x; y\rangle = [\![V]\!]\ \mathbf{in}\ [\![N]\!] \\
[\![\mathbf{case}\ V\ \{\ell\ x \mapsto M; y \mapsto N\}]\!] &= \mathbf{case}\ [\![V]\!]\ \{\ell\ x \mapsto [\![M]\!]; y \mapsto [\![N]\!]\} \\
[\![\mathbf{absurd}\ V]\!] &= \mathbf{absurd}\ [\![V]\!] \\
[\![\mathbf{return}\ V]\!] &= \lambda k.k\ [\![V]\!] \\
[\![\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N]\!] &= \lambda k.[\![M]\!](\lambda x.[\![N]\!]k)
\end{array}
$$

Figure 5 First-order CPS translation of fine-grain call-by-value.

4.1 CPS translation for fine-grain call-by-value

We start by giving a CPS translation of the handler-free subset of $\lambda^{\rho}_{\text{eff}}$ in Figure 5. Fine-grain call-by-value admits a particularly simple CPS translation due to the separation of values and computations. All constructs from the source language are translated homomorphically into the target language, except for **return**, **let**, and type abstraction (the translation performs type erasure). Lifting a value V to a computation **return** V is interpreted by passing the value to the current continuation. Sequencing two computations with **let** is translated in the usual continuation passing way. In addition, we explicitly η-expand the translation of a type abstraction in order to ensure that value terms in the source calculus translate to value terms in the target.

4.2 First-order CPS translations of handlers

As is usual for CPS, the translation of a computation term by the basic CPS translation takes a single continuation parameter that represents the context. With effects and handlers in the source language, we must now keep track of two kinds of context in which each computation executes: a *pure context* that tracks the state of pure computation in the scope of the current handler, and an *effect context* that describes how to handle operations in the scope of the current handler. Correspondingly, we have both *pure continuations* (k) and *effect continuations* (h). As handlers can be nested, each computation executes in the context of a *stack* of pairs of pure and effect continuations (as in the abstract machine of [11]).

On invocation of a handler, the pure continuation is initialised to a representation of the return clause and the effect continuation to a representation of the operation clauses. As pure computation proceeds, the pure continuation may grow. If an operation is encountered then

the effect continuation is invoked. The current continuation pair (k, h) is packaged up as a *resumption* and passed to the current handler along with the operation and its argument. The effect continuation then either handles the operation, invoking the resumption as appropriate, or forwards the operation to an outer handler. In the latter case, the resumption is modified to reinstate the dynamic stack of continuations when invoked.

The translations introduced in this subsection differ in how they represent stacks of pure and effect continuations, and how they represent resumptions. The first translation represents the stack of continuations using currying, and resumptions as functions (§4.2.1). Currying obstructs proper tail-recursion, so we move to an explicit representation of the stack (§4.2.2). Then, in order to avoid administrative reductions in our final higher-order one-pass translation we use an explicit representation of resumptions (§4.2.3).

4.2.1 Curried translation

Our first translation builds upon the CPS translation of Figure 5. The extension to operations and handlers is modest since currying conveniently lets us get away with a shift in interpretation: rather than accepting a single continuation, translated computation terms now accept an arbitrary even number of curried arguments representing the stack of pure and effect continuations. Thus, the translation of core constructs remain exactly the same as in Figure 5, where we imagine there being some number of extra continuation arguments that have been η-reduced. The translation of operations, handlers, and top-level programs is as follows.

$$
\begin{aligned}
[\![\mathbf{do}\ \ell\ V]\!] &= \lambda k.\lambda h.h\ (\ell\ \langle [\![V]\!], \lambda x.k\ x\ h\rangle)\\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= [\![M]\!]\ [\![H^{\mathrm{ret}}]\!]\ [\![H^{\mathrm{ops}}]\!],\ \text{where}\\
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x.\lambda h.[\![N]\!]\\
[\![\{\ell\ p\ r \mapsto N_\ell\}_{\ell\in\mathcal{L}}]\!] &= \lambda z.\mathbf{case}\ z\ \{(\ell\ \langle p, r\rangle \mapsto [\![N_\ell]\!])_{\ell\in\mathcal{L}}; y \mapsto M_{\mathrm{forward}}\}\\
M_{\mathrm{forward}} &= \lambda k'.\lambda h'.\mathbf{vmap}\,(\lambda\langle p, r\rangle\ k.k\,\langle p, \lambda x.r\,x\,k'\ h'\rangle)\,y\,h'\\
\top[\![M]\!] &= [\![M]\!]\ (\lambda x.\lambda h.x)\ (\lambda z.\mathbf{absurd}\ z)
\end{aligned}
$$

We extend our target calculus in order to implement forwarding. The computation term $\mathbf{vmap}\ U\ V\ W$ maps the function U over the body of the variant V and passes the result to continuation W. Its reduction rule is:

$$\text{U-VMap}\quad \mathbf{vmap}\,U\,(\ell\ V)\ W \rightsquigarrow U\ V\,(\lambda x\,k.k\,(\ell\,x))\ W$$

In an untyped setting **vmap** is easily definable. In Appendix B we sketch how to adapt our row type system to type the CPS translation with **vmap**.

The translation of $\mathbf{do}\ \ell\ V$ accepts a pure continuation k and an effect continuation h, which acts as a dispatcher function for encoded operations. Each operation is encoded as a value tagged with the name ℓ, where the value consists of a pair consisting of the parameter of the operation, and a resumption, which ensures that any subsequent operations are handled by the same effect continuation h.

The translation of $\mathbf{handle}\ M\ \mathbf{with}\ H$ invokes the translation of M with new pure and effect continuation arguments for the return and operation clauses of H. The translation of a return clause is a term which garbage collects the current effect continuation h. The translation of a set of operation clauses is a function which dispatches on encoded operations, and in the default case forwards to an outer handler. In the forwarding case, the resumption is extended by the parent continuation pair in order to reinstate the handler stack, thereby ensuring subsequent invocations of the same operation are handled uniformly.

Conceptually, top-level programs are enclosed by a top-level handler with an empty collection of operation clauses and an identity return clause. The CPS translation materialises

this handler as the identity pure continuation (the K combinator), and an effect continuation that is never intended to be called.

There are two practical problems with this initial translation. First, it is not properly tail-recursive due to the curried representation of the continuation stack. We will rectify this using an uncurried representation in the next subsection. Second, it yields administrative redexes. We will rectify this with a higher-order one-pass translation in §4.3.

To illustrate both issues, consider the following example:

$$\begin{aligned}\top[\![\mathbf{return}\ \langle\rangle]\!] &= (\lambda k.k\,\langle\rangle)\,(\lambda x.\lambda h.x)\,(\lambda z.\mathbf{absurd}\ z)\\ &\rightsquigarrow ((\lambda x.\lambda h.x)\,\langle\rangle)\,(\lambda z.\mathbf{absurd}\ z) \rightsquigarrow (\lambda h.\langle\rangle)\,(\lambda z.\mathbf{absurd}\ z) \rightsquigarrow \langle\rangle\end{aligned}$$

The first reduction is administrative: it has nothing to do with the dynamic semantics of the original term and there is no reason not to eliminate it statically. The second and third reductions simulate handling **return** $\langle\rangle$ at the top level. The second reduction partially applies $\lambda x.\lambda h.x$ to $\langle\rangle$, which must return a value so that the third reduction can be applied: evaluation is not tail-recursive. The lack of tail-recursion is also apparent in our relaxation of fine-grain call-by-value in Figure 4: the function position of an application can be a computation, and the calculus makes use of evaluation contexts.

Remark. We originally derived the curried CPS translation for effect handlers by composing a translation from effect handlers to delimited continuations [10] with a CPS translation for delimited continuations [22].

4.2.2 Uncurried translation: continuations as explicit stacks

Following Materzok and Biernacki [22] we uncurry our CPS translation in order to obtain a properly tail-recursive translation. The translation of return, let binding, operations, handlers, and top level programs is as follows.

$$\begin{aligned}[\![\mathbf{return}\ V]\!] &= \lambda(k :: ks).k\ [\![V]\!]\ ks\\ [\![\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N]\!] &= \lambda(k :: ks).[\![M]\!]((\lambda x\,ks.[\![N]\!](k :: ks)) :: ks)\\ [\![\mathbf{do}\ \ell\ V]\!] &= \lambda(k :: h :: ks).h\ (\ell\ \langle [\![V]\!], \lambda x\,ks.k\ x\ (h :: ks)\rangle)\ ks\\ [\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= \lambda ks.[\![M]\!]([\![H^{\mathrm{ret}}]\!] :: [\![H^{\mathrm{ops}}]\!] :: ks),\ \text{where}\\ [\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x\,ks.\mathbf{let}\ (h :: ks') = ks\ \mathbf{in}\ [\![N]\!]\ ks\\ [\![\{\ell\ p\ r \mapsto N_\ell\}_{\ell\in\mathcal{L}}]\!] &= \lambda z\,ks.\mathbf{case}\ z\ \{(\ell\ \langle p, r\rangle \mapsto [\![N_\ell]\!]\ ks)_{\ell\in\mathcal{L}};\ y \mapsto M_{\mathrm{forward}}\}\\ M_{\mathrm{forward}} &= \mathbf{let}\ (k' :: h' :: ks') = ks\ \mathbf{in}\\ &\qquad \mathbf{vmap}\,(\lambda\langle p, r\rangle\,(k :: ks).k\,\langle p, \lambda x\,ks.r\,x\,(k' :: h' :: ks)\rangle\,ks)\,y\,ks'\\ \top[\![M]\!] &= [\![M]\!]\ ((\lambda x\,ks.x) :: (\lambda z\,ks.\mathbf{absurd}\ z) :: [])\end{aligned}$$

The other cases are as in Figure 5. The stacks of continuations are now explicitly represented as lists, where pure continuations and effect continuations occupy alternating positions. We now require lists in our target, which we implement using right-nested pairs and unit:

$$[] \equiv \langle\rangle \qquad V :: W \equiv \langle V, W\rangle \qquad U :: V :: W \equiv \langle U, \langle V, W\rangle\rangle$$

Similarly, we extend pattern matching in the standard way to accommodate lists.

Since we now use a list representation for the stacks of continuations, we need to modify the translations of all the constructs that manipulate continuations. For **return** and **let**, we extract the top continuation k and manipulate it analogously to the original translation in Figure 5. For **do**, we extract the top pure continuation k and effect continuation h and invoke h in the same way as the curried translation, except that we explicitly maintain the stack ks of additional continuations. The translation of **handle**, however, pushes a continuation pair

onto the stack instead of supplying them as arguments. Handling of operations is the same as before, except for explicit passing of the ks. Forwarding now pattern matches on the stack to extract the next continuation pair, rather than accepting them as arguments. As we now use lists to represent stacks, we must modify the reduction rule for **vmap**.

U-VMap $\quad \mathbf{vmap}\ U\,(\ell\,V)\ W \rightsquigarrow U\ V\,(\lambda x\,(k :: ks).k\,(\ell\,x)\ W)$

Proper tail recursion coincides with a refinement of the target syntax. Now applications are either of the form $V\ W$ or of the form $U\ V\ W$. We could also add a rule for applying a two argument lambda abstraction to two arguments at once and eliminate the U-Lift rule, but we defer spelling out the details until §4.3.

4.2.3 Resumptions as explicit reversed stacks

In our two CPS translations so far, resumptions have been represented as functions and forwarding has been implemented by function composition. In order to avoid administrative reductions due to function composition, we move to an explicit representation of resumptions as *reversed* stacks of pure and effect continuations. We convert these reversed stacks to actual functions on demand using a special **fun** binding with the following reduction rule.

U-Fun $\quad \mathbf{let}\ r = \mathbf{fun}\,(V_n :: \ldots :: V_1 :: [])\ \mathbf{in}\ N \rightsquigarrow N[(\lambda x\,ks.V_1\,x\,(V_2 :: \ldots :: V_n :: ks))/r]$

This reduction rule reverses the stack, pulls out the top continuation V_1, and appends the remainder onto the current stack ks. The stack representing a resumption and the remaining stack ks are reminiscent of the zipper data structure for representing cursors in lists [12]. Resumptions represent pointers into the stack of handlers. We use exactly the same representation in our abstract machine for effect handlers [11].

The translations of **do**, handling, and forwarding need to be modified to handle the change in representation of resumptions. The translation of **do** builds a resumption stack, handling uses the **fun** construct to convert the resumption stack into a function, and M_{forward} extends the resumption stack with the current continuation pair.

$$
\begin{aligned}
[\![\mathbf{do}\ \ell\ V]\!] &= \lambda k :: h :: ks.h\,(\ell\ \langle [\![V]\!], h :: k :: []\rangle)\ ks \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell\in\mathcal{L}}\}]\!] &= \lambda z\,ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \mathbf{let}\ r = \mathbf{fun}\ s\ \mathbf{in}\ [\![N_\ell]\!]\ ks)_{\ell\in\mathcal{L}};\ y \mapsto M_{\text{forward}}\} \\
M_{\text{forward}} &= \mathbf{let}\ (k' :: h' :: ks') = ks\ \mathbf{in} \\
&\quad\ \mathbf{vmap}\,(\lambda\langle p, r\rangle\,(k :: ks).k\,\langle p, h' :: k' :: r\rangle\ ks)\ y\ ks'
\end{aligned}
$$

4.3 A higher-order explicit stack translation

We now adapt our uncurried CPS translation to a higher-order one-pass CPS translation [6] that partially evaluates administrative redexes at translation time. Following Danvy and Nielsen [7], we adopt a two-level lambda calculus notation to distinguish between *static* lambda abstraction and application in the meta language and *dynamic* lambda abstraction and application in the target language. The idea is that redexes marked as static are reduced as part of the translation (at compile time), whereas those marked as dynamic are reduced at runtime. The CPS translation is given in Figure 6.

An overline denotes a static syntax constructor and an underline denotes a dynamic syntax constructor. In order to facilitate this notation we write application explicitly as an infix "at" symbol (@). We assume the meta language is pure and hence respects the usual β and η equivalences. We extend the overline and underline notation to distinguish between static and dynamic let bindings.

The reify operator $\downarrow$ maps static lists to dynamic ones and the reflect constructor $\uparrow$ allows dynamic lists to be treated as static. We use list pattern matching in the meta language.

$$(\overline{\lambda}(\kappa :: \mathcal{P}).\mathcal{M})\ \overline{@}\ (V :: VS) = \overline{\mathbf{let}}\ \kappa = V\ \overline{\mathbf{in}}\ (\overline{\lambda}\mathcal{P}.\mathcal{M})\ \overline{@}\ VS$$
$$(\overline{\lambda}(\kappa :: \mathcal{P}).\mathcal{M})\ \overline{@}\ \uparrow V = \underline{\mathbf{let}}\ (k :: ks) = V\ \underline{\mathbf{in}}\ \overline{\mathbf{let}}\ \kappa = k\ \overline{\mathbf{in}}\ (\overline{\lambda}\mathcal{P}.\mathcal{M})\ \overline{@}\ \uparrow ks$$

Here we let $\mathcal{M}$ range over meta language expressions.

Now the target calculus is refined so that all lambda abstractions and applications take two arguments, the U-Lift rule is removed, and the U-App rule is replaced by the following reduction rule:

U-AppTwo $\quad (\underline{\lambda}x\ ks.\ M)\ \underline{@}\ V\ \underline{@}\ W \rightsquigarrow M[V/x, W/ks]$

We add an extra dummy argument to the translation of type lambda abstractions and applications in order to ensure that all dynamic functions take exactly two arguments. The single argument lambdas and applications from the first-order uncurried translation are still present, but now they are all static.

In order to reason about the behaviour of the S-Handle-op rule, which is defined in terms of an evaluation context, we extend the CPS translation to evaluation contexts:

$$\begin{aligned}
[\![[\ ]]\!] &= \quad \overline{\lambda}\kappa s.\ \kappa s \\
[\![\mathbf{let}\ x \leftarrow \mathcal{E}\ \mathbf{in}\ N]\!] &= \overline{\lambda}\kappa :: \kappa s.\ [\![\mathcal{E}]\!]\ \overline{@}\ ((\underline{\lambda}x\ ks.[\![N]\!]\ \overline{@}\ (k :: \uparrow ks)) :: \kappa s) \\
[\![\mathbf{handle}\ \mathcal{E}\ \mathbf{with}\ H]\!] &= \quad \overline{\lambda}\kappa s.\ [\![\mathcal{E}]\!]\ \overline{@}\ ([\![H^{\mathrm{ret}}]\!] :: [\![H^{\mathrm{ops}}]\!] :: \kappa s)
\end{aligned}$$

The following lemma is the characteristic property of the CPS translation on evaluation contexts. This allows us to focus on the computation contained within an evaluation context.

▶ **Lemma 3** (Decomposition). $[\![\mathcal{E}[M]]\!]\ \overline{@}\ (V :: VS) = [\![M]\!]\ \overline{@}\ ([\![\mathcal{E}]\!]\ \overline{@}\ (V :: VS))$.

Though we have eliminated the static administrative redexes, we are still left with one form of administrative redex that cannot be eliminated statically because it only appears at run-time. These arise from pattern matching against a reified stack of continuations and are given by the U-SplitList rule.

U-SplitList $\quad \underline{\mathbf{let}}\ (k :: ks) = V :: W\ \underline{\mathbf{in}}\ M \rightsquigarrow M[V/k, W/ks]$

This is isomorphic to the U-Split rule, but we now treat lists and U-SplitList as distinct from pairs, unit, and U-Split in the higher-order translation so that we can properly account for administrative reduction. We write $\rightsquigarrow_{\mathrm{a}}$ for the compatible closure of U-SplitList.

By definition, $\downarrow\uparrow V = V$, but we also need to reason about the inverse composition. The proof is by induction on the structure of M.

▶ **Lemma 4** (Reflect after reify). $[\![M]\!]\ \overline{@}\ (V_1 :: \ldots V_n :: \uparrow\downarrow VS) \rightsquigarrow_a^* [\![M]\!]\ \overline{@}\ (V_1 :: \ldots V_n :: VS)$.

We next observe that the CPS translation simulates forwarding.

▶ **Lemma 5** (Forwarding). *If $\ell \notin dom(H_1)$ then:*

$$[\![H_1^{\mathrm{ops}}]\!]\ \underline{@}\ \ell\ \langle U, V\rangle\ \underline{@}\ (V_2 :: [\![H_2^{\mathrm{ops}}]\!] :: W) \rightsquigarrow^+ [\![H_2^{\mathrm{ops}}]\!]\ \underline{@}\ \ell\ \langle U, [\![H_1^{\mathrm{ops}}]\!] :: V_1 :: V\rangle\ \underline{@}\ W\,.$$

Now we show that the translation simulates the S-Handle-op rule.

▶ **Lemma 6** (Handling). *If $\ell \notin BL(\mathcal{E})$ and $H^\ell = \{\ell\ p\ r \mapsto N_\ell\}$ then:*

$$\begin{aligned}
&[\![\mathbf{do}\ \ell\ V]\!]\ \overline{@}\ ([\![\mathcal{E}]\!]\ \overline{@}\ ([\![H^{\mathrm{ret}}]\!] :: [\![H^{\mathrm{ops}}]\!] :: VS)) \rightsquigarrow^+ \rightsquigarrow_a^* \\
&\quad ([\![N_\ell]\!]\ \overline{@}\ VS)[[\![V]\!]/p, (\lambda y\ ks.[\![\mathbf{return}\ y]\!]\ \overline{@}\ ([\![\mathcal{E}]\!]\ \overline{@}\ ([\![H^{\mathrm{ret}}]\!] :: [\![H^{\mathrm{ops}}]\!] :: \uparrow ks)))/r]\,.
\end{aligned}$$

This lemma follows from Lemmas 3, 4, and 5. We now turn to our main result which is a simulation result in style of Plotkin [24]. The theorem shows that the only extra behaviour exhibited by a translated term is the bureaucracy of deconstructing the continuation stack.

Static patterns and static lists

$$\begin{array}{lll} \text{Static patterns} & \mathcal{P} & ::= \kappa :: \mathcal{P} \mid \kappa s \\ \text{Static lists} & VS & ::= V :: VS \mid \uparrow V \end{array}$$

Reify

$$\begin{aligned} \downarrow(V :: VS) &= V :: \downarrow VS \\ \downarrow\uparrow V &= V \end{aligned}$$

Values

$$\begin{aligned} [\![x]\!] &= x \\ [\![\lambda x.M]\!] &= \lambda x\, ks.[\![M]\!] \mathbin{@} \uparrow ks \\ [\![\Lambda\alpha.M]\!] &= \lambda z\, ks.[\![M]\!] \mathbin{@} \uparrow ks \\ [\![\langle\rangle]\!] &= \langle\rangle \\ [\![\langle \ell = V; W\rangle]\!] &= \langle \ell = [\![V]\!]; [\![W]\!]\rangle \\ [\![\ell\ V]\!] &= \ell\ [\![V]\!] \end{aligned}$$

Computations

$$\begin{aligned} [\![V\ W]\!] &= \overline{\lambda}\kappa s.[\![V]\!] \mathbin{@} [\![W]\!] \mathbin{@} \downarrow\kappa s \\ [\![V\ T]\!] &= \overline{\lambda}\kappa s.[\![V]\!] \mathbin{@} \langle\rangle \mathbin{@} \downarrow\kappa s \\ [\![\mathbf{let}\ \langle \ell = x; y\rangle = V\ \mathbf{in}\ N]\!] &= \overline{\lambda}\kappa s.\mathbf{let}\ \langle \ell = x; y\rangle = [\![V]\!]\ \mathbf{in}\ [\![N]\!] \mathbin{@} \kappa s \\ [\![\mathbf{case}\ V\ \{\ell\ x \mapsto M; y \mapsto N\}]\!] &= \overline{\lambda}\kappa s.\mathbf{case}\ [\![V]\!]\ \{\ell\ x \mapsto [\![M]\!] \mathbin{@} \kappa s; y \mapsto [\![N]\!] \mathbin{@} \kappa s\} \\ [\![\mathbf{absurd}\ V]\!] &= \overline{\lambda}\kappa s.\mathbf{absurd}\ [\![V]\!] \\ [\![\mathbf{return}\ V]\!] &= \overline{\lambda}\kappa :: \kappa s.\kappa \mathbin{@} [\![V]\!] \mathbin{@} \downarrow\kappa s \\ [\![\mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N]\!] &= \overline{\lambda}\kappa :: \kappa s.[\![M]\!] \mathbin{@} ((\lambda x\, ks.[\![N]\!] \mathbin{@} (\kappa :: \uparrow ks)) :: \kappa s) \\ [\![\mathbf{do}\ \ell\ V]\!] &= \overline{\lambda}\kappa :: \eta :: \kappa s.\eta \mathbin{@} (\ell\ \langle [\![V]\!], \eta :: \kappa :: []\rangle) \mathbin{@} \downarrow\kappa s \\ [\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= \overline{\lambda}\kappa s.[\![M]\!] \mathbin{@} ([\![H^{\mathrm{ret}}]\!] :: [\![H^{\mathrm{ops}}]\!] :: \kappa s),\ \text{where} \\ [\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \lambda x\, ks.\mathbf{let}\ (h :: ks') = ks\ \mathbf{in}\ [\![N]\!] \mathbin{@} \uparrow ks' \\ [\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell\in\mathcal{L}}\}]\!] &= \lambda z\, ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \mathbf{let}\ r = \mathbf{fun}\ s\ \mathbf{in}\ [\![N_\ell]\!] \mathbin{@} \uparrow ks)_{\ell\in\mathcal{L}};\ y \mapsto M_{\mathrm{forward}}\} \\ M_{\mathrm{forward}} &= \mathbf{let}\ (k' :: h' :: ks') = ks\ \mathbf{in} \\ &\quad \mathbf{vmap}\ (\lambda\langle p, s\rangle\ (k :: ks).k\ \langle p, h' :: k' :: s\rangle\ ks)\ y\ ks' \end{aligned}$$

Top level program

$$\top[\![M]\!] = [\![M]\!] \mathbin{@} ((\lambda x\, ks.x) :: (\lambda z\, ks.\mathbf{absurd}\ z) :: \uparrow[])$$

Figure 6 Higher-order uncurried CPS translation of $\lambda^{\rho}_{\mathrm{eff}}$.

▶ **Theorem 7** (Simulation). *If $M \rightsquigarrow N$ then $\top[\![M]\!] \rightsquigarrow^{+} \rightsquigarrow_a^{*} \top[\![N]\!]$.*

The proof is by case analysis on the reduction relation using Lemmas 3–6. The S-Handle-op case follows from Lemma 6.

In common with most CPS translations, full abstraction does not hold. However, as our semantics is deterministic it is straightforward to show a backward simulation result.

▶ **Corollary 8** (Backwards simulation). *If $\top[\![M]\!] \rightsquigarrow^{+} \rightsquigarrow_a^{*} V$ then there exists W such that $M \rightsquigarrow^{*} W$ and $\top[\![W]\!] = V$.*

4.4 Shallow handlers

Shallow handlers [13] differ from deep handlers in that when handling an operation the former does not reinvoke the handler inside the resumption. The typing rules and operational semantics for shallow handlers are as follows.

T-Shallow-Handle

$$\frac{\Delta; \Gamma \vdash M : C \qquad \Delta; \Gamma \vdash H : C \Rightarrow^{\dagger} D}{\Delta; \Gamma \vdash \mathbf{handle}^{\dagger}\ M\ \mathbf{with}\ H : D}$$

T-Shallow-Handler

$$\frac{\begin{array}{c} C = A!\{(\ell_i : A_i \to B_i)_i; R\} \qquad D = B!\{(\ell_i : P_i)_i; R\} \\ H = \{\mathbf{return}\ x \mapsto M\} \uplus \{\ell_i\ y\ r \mapsto N_{\ell_i}\}_i \\ \Delta; \Gamma, x : A \vdash M : D \\ [\Delta; \Gamma, y : A_i, r : B_i \to C \vdash N_{\ell_i} : D]_i \end{array}}{\Delta; \Gamma \vdash H : C \Rightarrow^{\dagger} D}$$

S-Shallow-Ret $\quad \mathbf{handle}^{\dagger}\ (\mathbf{return}\ V)\ \mathbf{with}\ H \rightsquigarrow N[V/x]$, where $H^{\mathrm{ret}} = \{\mathbf{return}\ x \mapsto N\}$

S-Shallow-Op $\quad \mathbf{handle}^{\dagger}\ \mathcal{E}[\mathbf{do}\ \ell\ V]\ \mathbf{with}\ H \rightsquigarrow N[V/x, (\lambda y.\mathcal{E}[\mathbf{return}\ y])/r]$, where $\ell \notin BL(\mathcal{E})$ and $H^{\ell} = \{\ell\ x\ r \mapsto N\}$

We write a dagger (†) superscript to distinguish shallow handlers from deep handlers. We adapt our higher-order CPS translation to accommodate both shallow and deep handlers.

$$\left.\begin{aligned}
[\![\mathbf{do}\ \ell\ V]\!] &= \overline{\lambda}\kappa\, \overline{::}\, \eta\, \overline{::}\, \kappa s.\eta\ \underline{@}\ (\ell\ \langle [\![V]\!], \kappa\ \underline{::}\ []\rangle)\ \underline{@}\ \downarrow\kappa s \\
[\![\mathbf{handle}\ M\ \mathbf{with}\ H]\!] &= \overline{\lambda}\kappa s.[\![M]\!]\ \overline{@}\ ([\![H^{\mathrm{ret}}]\!]\ \overline{::}\ [\![H^{\mathrm{ops}}]\!]\ \overline{::}\ \kappa s) \\
[\![\mathbf{handle}^{\dagger}\ M\ \mathbf{with}\ H]\!] &= \overline{\lambda}\kappa s.[\![M]\!]\ \overline{@}\ ([\![H^{\mathrm{ret}}]\!]\ \overline{::}\ [\![H^{\mathrm{ops}}]\!]^{\dagger}\ \overline{::}\ \kappa s)
\end{aligned}\right\} \text{where}$$

$$\begin{aligned}
[\![\{\mathbf{return}\ x \mapsto N\}]\!] &= \underline{\lambda} x\ ks.\underline{\mathbf{let}}\ (h\ \underline{::}\ ks') = ks\ \underline{\mathbf{in}}\ [\![N]\!]\ \overline{@}\uparrow ks' \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell\in\mathcal{L}}\}]\!] &= \mathbf{rec}\, h\, z\, ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \underline{\mathbf{let}}\ r = \mathbf{fun}\,(h\ \underline{::}\ s)\ \underline{\mathbf{in}}\ [\![N_\ell]\!]\ \overline{@}\uparrow ks)_{\ell\in\mathcal{L}} \\
&\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
[\![\{(\ell\ p\ r \mapsto N_\ell)_{\ell\in\mathcal{L}}\}]\!]^{\dagger} &= \mathbf{rec}\, h\, z\, ks.\mathbf{case}\ z\ \{(\ell\ \langle p, s\rangle \mapsto \underline{\mathbf{let}}\ r = \mathbf{fun}\,(V_{\mathrm{id}}\ \underline{::}\ s)\ \underline{\mathbf{in}}\ [\![N_\ell]\!]\ \overline{@}\uparrow ks)_{\ell\in\mathcal{L}} \\
&\qquad\qquad y \mapsto M_{\mathrm{forward}}\} \\
M_{\mathrm{forward}} &= \underline{\mathbf{let}}\ (k'\ \underline{::}\ h'\ \underline{::}\ ks') = ks\ \underline{\mathbf{in}} \\
&\qquad \mathbf{vmap}\ (\underline{\lambda}\langle p, s\rangle\ (k\ \underline{::}\ ks).k\,\langle p, h'\ \underline{::}\ k'\ \underline{::}\ h\ \underline{::}\ s\rangle\ ks)\ y\ ks' \\
V_{\mathrm{id}} &= \mathbf{rec}\, h\, y\, ks.M_{\mathrm{forward}}
\end{aligned}$$

For deep handlers, the current effect continuation is now added inside the translation of the operation clauses rather than the translation of the operation. This necessitates making the translation of operation clauses recursive, which we do using a recursion operator.

U-Rec $\quad (\mathbf{rec}\, f\, x\, ks.M)\ V\ W \rightsquigarrow M[(\mathbf{rec}\, f\, x.M)/f, V/x, W/ks]$

In order to translate a shallow handler we insert an identity effect continuation in place of the current effect continuation.

4.5 Exceptions and their handlers as separate constructs

Our core calculus $\lambda^{\rho}_{\mathrm{eff}}$ (and also Links) does not have special support for exceptions and their handlers. Indeed, exceptions are a special case of effects where the operations return an uninhabited type similar to the `Fail` operation from §2. On the other hand, Multicore OCaml maintains effects separate from exceptions not only for backwards compatibility but also due to the fact that exceptions in Multicore OCaml are cheaper than effects. Multicore OCaml relies on runtime support for stack manipulation in order to implement effect handlers, where raising an exception need only unwind the stack and need not capture the continuation. Thus, there is benefit in separating exceptions from effects; computations which raise exceptions but do not perform other effects may be retained in direct-style in a selective CPS translation. We can model this by extending handlers with exception clauses.

Handlers $\quad H ::= \{\mathbf{return}\ x \mapsto M\} \mid \{\ell\ x\ r \mapsto M\} \uplus H \mid \{\ell\ x \mapsto M\} \uplus H$

S-Handle-Ex $\quad \mathbf{handle}\ \mathcal{E}[\mathbf{raise}\ \ell\ V]\ \mathbf{with}\ H \rightsquigarrow N[V/x]$, where $\ell \notin BL(\mathcal{E})$ and $H^{\ell} = \{\ell\ x \mapsto N\}$

Exception clauses lack a resumption. The CPS translation can now be adapted to maintain a stack of triples (pure continuation, exception continuation, effect continuation).

5 Implementation

In this section, we briefly outline our experiences of implementing the CPS translations described in §4.

A first prototype (available at `https://github.com/Armael/lam`) was implemented by the fourth author and Armaël Guéneau for Multicore OCaml, which makes a distinction between exception handlers and general effect handlers. Thus, this implementation is based on the approach of §4.5. It uses a curried CPS translation which is not properly tail-recursive.

The Links implementation (available at `https://github.com/links-lang/links`) relies on the higher-order CPS translation of §4.3. The Links client-side JavaScript backend has

long used a higher-order CPS translation, relying on a trampoline for supporting lightweight concurrency and responsive user-interfaces. As it is based on a trampoline, the stack is periodically discarded and it is essential that the CPS translation be properly tail-recursive. Initially we attempted to implement a higher-order curried translation. We then realised that it is unclear whether it is even possible to define a higher-order curried translation for effect handlers, so we began implementing a first-order curried translation. It quickly became apparent that this approach could not work given the need to be properly tail-recursive. At this point we changed tack and successfully implemented a properly tail-recursive higher-order uncurried translation along the lines of the one described in §4.3.

6 Conclusions and future work

We have carried out a comprehensive study of CPS translations for effect handlers. We have presented the first full CPS translations for effect handlers: our translations go all the way to lambda calculus without relying on a special low-level handling construct as Leijen [17] does. We began with a standard first-order call-by-value CPS translation, which we extended to support effect handlers. We then refined the first-order translation by uncurrying it in order to yield a properly tail-recursive translation, and by adapting it to a higher-order one-pass translation that statically eliminates administrative redexes. We proved that the higher-order uncurried CPS translation simulates reduction in the source language. In addition, we have also shown how to adapt the translations to support shallow handlers.

The server backend for Links [11] is based on an extension of a CEK machine to support handlers. There are clear connections between their abstract machine and our higher-order CPS translation. In future work we intend to make these connections precise.

While our translations apply to unary handlers, we would like to investigate how to adapt them to *multi handlers* which handle multiple computations at simultaneously [20].

Many useful effect handlers do not use resumptions more than once. The Multicore OCaml compiler takes advantage of supporting only affine use of resumptions, by default, to obtain remarkably strong performance. However, Multicore OCaml makes use of its own custom stack implementation, whereas for certain backends (notably JavaScript) that luxury is not available. We would like to explore linear and affine variants of our CPS translations, mediated by a suitable substructural type system, to see if we can obtain similar benefits. Another direction we intend to explore in this setting is the use of JavaScript's existing generator abstraction for implementing linear and affine handlers. We also intend to perform a quantitative study of implementation strategies for effect handlers.

Acknowledgements. We thank the anonymous reviewers for their insightful comments.

References

1 Andrew W. Appel. *Compiling with Continuations*. Cambridge University Press, 1992.
2 Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. *J. Log. Algebr. Meth. Program.*, 84(1):108–123, 2015.
3 Bernard Berthomieu and Camille le Moniès de Sagazan. A calculus of tagged types, with applications to process languages. In *Workshop on Types for Program Analysis*, 1995.
4 Edwin Brady. Programming and reasoning with algebraic effects and dependent types. In *ICFP*, pages 133–144. ACM, 2013.
5 Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop. Links: Web programming without tiers. In *FMCO*, volume 4709 of *LNCS*, pages 266–296. Springer, 2006.

6 Olivier Danvy and Andrzej Filinski. Abstracting control. In *LISP and Functional Programming*, pages 151–160, 1990.
7 Olivier Danvy and Lasse R. Nielsen. A first-order one-pass CPS transformation. *Theor. Comput. Sci.*, 308(1-3):239–257, 2003.
8 Stephen Dolan, Leo White, K. C. Sivaramakrishnan, Jeremy Yallop, and Anil Madhavapeddy. Effective concurrency through algebraic effects. OCaml Workshop, 2015.
9 Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations. In *PLDI*, pages 237–247. ACM, 1993.
10 Yannick Forster, Ohad Kammar, Sam Lindley, and Matija Pretnar. On the expressive power of user-defined effects: Effect handlers, monadic reflection, delimited control. *Proc. ACM Program. Lang.*, 1(ICFP), September 2017.
11 Daniel Hillerström and Sam Lindley. Liberating effects with rows and handlers. In *TyDe@ICFP*, pages 15–27. ACM, 2016.
12 Gérard P. Huet. The zipper. *J. Funct. Program.*, 7(5):549–554, 1997.
13 Ohad Kammar, Sam Lindley, and Nicolas Oury. Handlers in action. In *ICFP*, pages 145–158. ACM, 2013.
14 Andrew Kennedy. Compiling with continuations, continued. In *ICFP*, pages 177–190. ACM, 2007.
15 Oleg Kiselyov and Hiromi Ishii. Freer monads, more extensible effects. In *Haskell*, pages 94–105. ACM, 2015.
16 Oleg Kiselyov and KC Sivaramakrishnan. Eff directly in OCaml. ML Workshop, 2016.
17 Daan Leijen. Type directed compilation of row-typed algebraic effects. In *POPL*, pages 486–499. ACM, 2017.
18 Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Inf. Comput.*, 185(2):182–210, 2003.
19 Sam Lindley and James Cheney. Row-based effect types for database integration. In *TLDI*, pages 91–102. ACM, 2012.
20 Sam Lindley, Conor McBride, and Craig McLaughlin. Do be do be do. In *POPL*, pages 500–514. ACM, 2017.
21 Marek Materzok and Dariusz Biernacki. Subtyping delimited continuations. In *ICFP*, pages 81–93. ACM, 2011.
22 Marek Materzok and Dariusz Biernacki. A dynamic interpretation of the CPS hierarchy. In *APLAS*, volume 7705 of *LNCS*, pages 296–311. Springer, 2012.
23 Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.
24 Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.
25 Gordon D. Plotkin and John Power. Adequacy for algebraic effects. In *FoSSaCS*, volume 2030 of *LNCS*, pages 1–24. Springer, 2001.
26 Gordon D. Plotkin and Matija Pretnar. Handling algebraic effects. *Logical Methods in Computer Science*, 9(4), 2013. doi:10.2168/LMCS-9(4:23)2013.
27 Matija Pretnar. An introduction to algebraic effects and handlers. *Electr. Notes Theor. Comput. Sci.*, 319:19–35, 2015. Invited tutorial paper.
28 Didier Remy. Syntactic theories and the algebra of record terms. Technical Report RR-1869, INRIA, 1993.
29 Philip Wadler. The essence of functional programming. In *POPL*, pages 1–14. ACM, 1992.

A Kinding and typing rules for $\lambda^{\rho}_{\text{eff}}$

The kinding rules for $\lambda^{\rho}_{\text{eff}}$ are given in Figure 7 and the typing rules are given in Figure 8.

$$\textsc{TyVar}\ \frac{}{\Delta, \alpha : K \vdash \alpha : K} \qquad \textsc{Comp}\ \frac{\Delta \vdash A : \mathsf{Type} \quad \Delta \vdash E : \mathsf{Effect}}{\Delta \vdash A!E : \mathsf{Comp}} \qquad \textsc{Fun}\ \frac{\Delta \vdash A : \mathsf{Type} \quad \Delta \vdash C : \mathsf{Comp}}{\Delta \vdash A \to C : \mathsf{Type}} \qquad \textsc{Forall}\ \frac{\Delta, \alpha : K \vdash C : \mathsf{Comp}}{\Delta \vdash \forall \alpha^K.\, C : \mathsf{Type}}$$

$$\textsc{Record}\ \frac{\Delta \vdash R : \mathsf{Row}_{\emptyset}}{\Delta \vdash \langle R \rangle : \mathsf{Type}} \qquad \textsc{Variant}\ \frac{\Delta \vdash R : \mathsf{Row}_{\emptyset}}{\Delta \vdash [R] : \mathsf{Type}} \qquad \textsc{Effect}\ \frac{\Delta \vdash R : \mathsf{Row}_{\emptyset}}{\Delta \vdash \{R\} : \mathsf{Effect}}$$

$$\textsc{Present}\ \frac{\Delta \vdash A : \mathsf{Type}}{\Delta \vdash \mathsf{Pre}(A) : \mathsf{Presence}} \qquad \textsc{Absent}\ \frac{}{\Delta \vdash \mathsf{Abs} : \mathsf{Presence}} \qquad \textsc{EmptyRow}\ \frac{}{\Delta \vdash \cdot : \mathsf{Row}_{\mathcal{L}}} \qquad \textsc{ExtendRow}\ \frac{\Delta \vdash P : \mathsf{Presence} \quad \Delta \vdash R : \mathsf{Row}_{\mathcal{L} \uplus \{\ell\}}}{\Delta \vdash \ell : P; R : \mathsf{Row}_{\mathcal{L}}}$$

$$\textsc{Handler}\ \frac{\Delta \vdash C : \mathsf{Comp} \qquad \Delta \vdash D : \mathsf{Comp}}{\Delta \vdash C \Rightarrow D : \mathsf{Handler}}$$

Figure 7 Kinding rules for $\lambda^{\rho}_{\text{eff}}$.

Values

$$\textsc{T-Var}\ \frac{x : A \in \Gamma}{\Delta; \Gamma \vdash x : A} \qquad \textsc{T-Lam}\ \frac{\Delta; \Gamma, x : A \vdash M : C}{\Delta; \Gamma \vdash \lambda x^A.\, M : A \to C} \qquad \textsc{T-PolyLam}\ \frac{\Delta, \alpha : K; \Gamma \vdash M : C \qquad \alpha \notin FTV(\Gamma)}{\Delta; \Gamma \vdash \Lambda \alpha^K.\, M : \forall \alpha^K.\, C}$$

$$\textsc{T-Unit}\ \frac{}{\Delta; \Gamma \vdash \langle\rangle : \langle\rangle} \qquad \textsc{T-Extend}\ \frac{\Delta; \Gamma \vdash V : A \qquad \Delta; \Gamma \vdash W : \langle \ell : \mathsf{Abs}; R \rangle}{\Delta; \Gamma \vdash \langle \ell = V; W \rangle : \langle \ell : \mathsf{Pre}(A); R \rangle} \qquad \textsc{T-Inject}\ \frac{\Delta; \Gamma \vdash V : A}{\Delta; \Gamma \vdash (\ell\ V)^R : [\ell : \mathsf{Pre}(A); R]}$$

Computations

$$\textsc{T-App}\ \frac{\Delta; \Gamma \vdash V : A \to C \qquad \Delta; \Gamma \vdash W : A}{\Delta; \Gamma \vdash V\ W : C} \qquad \textsc{T-PolyApp}\ \frac{\Delta; \Gamma \vdash V : \forall \alpha^K.\, C \qquad \Delta \vdash T : K}{\Delta; \Gamma \vdash V\ T : C[T/\alpha]} \qquad \textsc{T-Split}\ \frac{\Delta; \Gamma \vdash V : \langle \ell : \mathsf{Pre}(A); R \rangle \qquad \Delta; \Gamma, x : A, y : \langle \ell : \mathsf{Abs}; R \rangle \vdash N : C}{\Delta; \Gamma \vdash \mathbf{let}\ \langle \ell = x; y \rangle = V\ \mathbf{in}\ N : C}$$

$$\textsc{T-Case}\ \frac{\Delta; \Gamma \vdash V : [\ell : \mathsf{Pre}(A); R] \qquad \Delta; \Gamma, x : A \vdash M : C \qquad \Delta; \Gamma, y : [\ell : \mathsf{Abs}; R] \vdash N : C}{\Delta; \Gamma \vdash \mathbf{case}\ V \{\ell\ x \mapsto M; y \mapsto N\} : C} \qquad \textsc{T-Absurd}\ \frac{\Delta; \Gamma \vdash V : []}{\Delta; \Gamma \vdash \mathbf{absurd}^C\ V : C}$$

$$\textsc{T-Return}\ \frac{\Delta; \Gamma \vdash V : A}{\Delta; \Gamma \vdash \mathbf{return}\ V : A!E} \qquad \textsc{T-Let}\ \frac{\Delta; \Gamma \vdash M : A!E \qquad \Delta; \Gamma, x : A \vdash N : B!E}{\Delta; \Gamma \vdash \mathbf{let}\ x \leftarrow M\ \mathbf{in}\ N : B!E}$$

$$\textsc{T-Do}\ \frac{\Delta; \Gamma \vdash V : A \qquad E = \{\ell : A \to B; R\}}{\Delta; \Gamma \vdash (\mathbf{do}\ \ell\ V)^E : B!E} \qquad \textsc{T-Handle}\ \frac{\Delta; \Gamma \vdash M : C \qquad \Delta; \Gamma \vdash H : C \Rightarrow D}{\Delta; \Gamma \vdash \mathbf{handle}\ M\ \mathbf{with}\ H : D}$$

Handlers

$$\textsc{T-Handler}\ \frac{\begin{array}{c} C = A!\{(\ell_i : A_i \to B_i)_i; R\} \qquad D = B!\{(\ell_i : P_i)_i; R\} \qquad H = \{\mathbf{return}\ x \mapsto M\} \uplus \{\ell_i\ y\ r \mapsto N_{\ell_i}\}_i \\ [\Delta; \Gamma, y : A_i, r : B_i \to D \vdash N_{\ell_i} : D]_i \qquad \Delta; \Gamma, x : A \vdash M : D \end{array}}{\Delta; \Gamma \vdash H : C \Rightarrow D}$$

Figure 8 Typing rules for $\lambda^{\rho}_{\text{eff}}$.

B Typed CPS translations

Given a suitably polymorphic target calculus, we can define a typed CPS translation for $\lambda^{\rho}_{\text{eff}}$. Of course, we can use the polymorphism of the target calculus to model polymorphism (including row polymorphism) in source terms. More importantly, polymorphism may be used to abstract over the return type of effectful computations.

Operations may be encoded with polymorphic variants and handlers with case expressions. Alas, our existing row type system is not quite expressive enough to encode generic forwarding in this setting, so let us begin by considering the restriction of $\lambda^{\rho}_{\text{eff}}$ without forwarding, where we assume all effect rows are closed and all handlers are complete (i.e. include operation clauses for all operations in their types).

B.1 Handlers without forwarding

Figure 9 gives the CPS translation of $\lambda^{\rho}_{\text{eff}}$ without forwarding. The image of the translation lies within the pure fragment of $\lambda^{\rho}_{\text{eff}}$. The translations on value types and values are omitted as they are entirely homomorphic. If we erase all types then we obtain the translation of §4.2.1 with two differences. First, the former translation η-expands the body of a type lambda in order to ensure that it is a value (this is a superficial difference). There is no need to do that here as type lambdas are values. Second, the former translation performs forwarding, which we address in §B.2.

As the translation on terms is in essence one we have already seen, the main interest is in the translation on types. The body of the translation of a computation type $\mathcal{C}_{A,E}$ is exactly that of the continuation monad instantiated with return type $(\llbracket E \rrbracket \gamma \to \gamma) \to \gamma$. The argument to this function is the type of a handler continuation whose eventual return type is γ. By abstracting over the type of γ we can allow the handler stack to grow dynamically as necessary. This polymorphism accomplishes a similar purpose to Materzok and Biernacki's subtyping system for delimited continuations [21], which allows arbitrarily nested delimited continuations to be typed. We can witness the instantiation of the return type in the translation of a handler where the translation of the computation being handled $\llbracket M \rrbracket$ is applied to $\mathcal{C}_{B,E'}$. Polymorphism allows us to dynamically construct an arbitrarily deep stack of continuation monads, each carrying its own handler continuation.

Effect types type operations, which are encoded as variants pairing up a value with a continuation. The translation on effect types is parameterised by return type C.

▶ **Theorem 9** (Type preservation). *If* $\Delta; \Gamma \vdash M : A!E$ *then* $\llbracket \Delta \rrbracket; \llbracket \Gamma \rrbracket \vdash \llbracket M \rrbracket : \llbracket A!B \rrbracket$.

B.2 Forwarding and shapes

In order to encode forwarding we need to be able to parametrically specify what a default case does. Given a default variable y, we know that y is of the form $\ell \langle V, W \rangle$ where $V : A$ and $W : B \to C$ for some unknown types A and B and a fixed return type C. From y we need to produce a new value $\ell \langle V, W' \rangle$ where $W' : B \to C'$ and C' is a new return type. We can do so if we can define a typed version of the **vmap** operation.

The extension we propose to our row type system is to allow a row type to be given a *shape*, also known as a *type scheme*, which constrains the form of the ordinary types it contains. For instance, the shape of a row for a variant representing an operation at return type C is $\alpha^{\mathsf{Type}}\, \beta^{\mathsf{Type}}.\langle \alpha, \beta \to C \rangle$ and the shape of an unconstrained row, that is the shape of all rows in plain $\lambda^{\rho}_{\text{eff}}$, is $\alpha^{\mathsf{Type}}.\alpha$.

Computation types

$$\mathcal{C}_{A,E} = (\llbracket A \rrbracket \to (\llbracket E \rrbracket \gamma \to \gamma) \to \gamma) \to (\llbracket E \rrbracket \gamma \to \gamma) \to \gamma$$
$$\llbracket A ! E \rrbracket = \forall \gamma^{\mathsf{Type}}. \mathcal{C}_{A,E}$$

Effect types

$$\llbracket \{\ell : \llbracket A_\ell \rrbracket \to \llbracket B_\ell \rrbracket\}_{\ell \in \mathcal{L}} \rrbracket C = [\ell : \langle \llbracket A_\ell \rrbracket, \llbracket B_\ell \rrbracket \to C \rangle]_{\ell \in \mathcal{L}}$$

Computations (the other cases are homomorphic)

$$\begin{aligned}
\llbracket (\mathbf{return}\ V)^{A!E} \rrbracket &= \Lambda \gamma^{\mathsf{Type}}. \lambda k^{\llbracket A \rrbracket \to (\llbracket E \rrbracket \gamma \to \gamma) \to \gamma}. k\ \llbracket V \rrbracket \\
\llbracket \mathbf{let}\ x \leftarrow M^{A!E}\ \mathbf{in}\ N^{B!E} \rrbracket &= \Lambda \gamma^{\mathsf{Type}}. \lambda k^{\llbracket B \rrbracket \to (\llbracket E \rrbracket \gamma \to \gamma) \to \gamma}. \llbracket M \rrbracket (\lambda x^{\llbracket B \rrbracket}. \llbracket N \rrbracket k) \\
\llbracket (\mathbf{do}\ \ell\ V)^{E} \rrbracket &= \Lambda \gamma^{\mathsf{Type}}. \lambda k^{\llbracket B \rrbracket \to (\llbracket E \rrbracket \gamma \to \gamma) \to \gamma}. \lambda h^{\llbracket E \rrbracket \gamma \to \gamma}. h\ (\ell\ \langle \llbracket V \rrbracket, \lambda x^{\llbracket B \rrbracket}. k\ x\ h \rangle)^{\llbracket E \rrbracket \gamma} \\
&\qquad \text{where } \ell : A \to B \in E \\
\llbracket \mathbf{handle}\ M\ \mathbf{with}\ H^{A!E \Rightarrow B!E'} \rrbracket &= \Lambda \gamma^{\mathsf{Type}}. \llbracket M \rrbracket\ \mathcal{C}_{B,E'}\ \llbracket H^{\mathrm{ret}} \rrbracket^{A!E \Rightarrow B!E'}\ \llbracket H^{\mathrm{ops}} \rrbracket^{A!E \Rightarrow B!E'}, \text{ where} \\
\llbracket \{\mathbf{return}\ x \mapsto N_{\mathrm{ret}}\} \rrbracket^{A!E \Rightarrow B!E'} &= \lambda x^{\llbracket A \rrbracket}. \lambda h^{\llbracket E \rrbracket \mathcal{C}_{B,E'} \to \mathcal{C}_{B,E'}}. \llbracket N_{\mathrm{ret}} \rrbracket \\
\llbracket \{\ell\ p\ r \mapsto N_\ell\}_{\ell \in \mathcal{L}} \rrbracket^{A!E \Rightarrow B!E'} &= \lambda z^{\llbracket E \rrbracket \mathcal{C}_{B,E'}}. \mathbf{case}\ z\ \{(\ell\ \langle p, r \rangle \mapsto \llbracket N_\ell \rrbracket \gamma)_{\ell \in \mathcal{L}}\}
\end{aligned}$$

Figure 9 Typed first-order curried CPS translation of $\lambda^{\rho}_{\mathrm{eff}}$ without forwarding.

With shapes we can give **vmap** the following typing rule

$$\text{SH-T-VMap}\quad \frac{\Delta \vdash R : \mathsf{Row}((\alpha_i^{K_i})_i.A, \emptyset) \qquad \Delta; \Gamma \vdash U : \forall (\alpha_i^{K_i})_i.A \to (B \to C) \to C \qquad \Delta; \Gamma \vdash V : [R] \qquad \Delta; \Gamma \vdash W : [((\alpha_i^{K_i})_i.A \Rightarrow B)R] \to C}{\Delta; \Gamma \vdash \mathbf{vmap}\ U\ V\ W : C}$$

where row kinds now take an additional shape parameter and the shape of a row type R of kind $\mathsf{Row}((\alpha_i^{K_i})_i.A, \mathcal{L})$ may be transformed using the special type operator $((\alpha_i^{K_i})_i.A \Rightarrow B)$ to a row of kind $\mathsf{Row}((\alpha_i^{K_i})_i.B, \mathcal{L})$. The CPS translation on the operation clauses now becomes

$$\llbracket \{\ell\ p\ r \mapsto N_\ell\}_{\ell \in \mathcal{L}} \rrbracket^{A!E \Rightarrow B!E'} = \lambda z^{\llbracket E \rrbracket \mathcal{C}_{B,E'}}. \mathbf{case}\ z\ \{(\ell\ \langle p, r \rangle \mapsto \llbracket N_\ell \rrbracket \gamma)_{\ell \in \mathcal{L}};\ y \mapsto M_{\mathrm{forward}}\}$$

where

$$\begin{aligned}
M_{\mathrm{forward}} &= \lambda k'^{\llbracket B \rrbracket \to C' \to \gamma}. \lambda h'^{C'}. \\
&\qquad \mathbf{vmap}\ (\Lambda \alpha^{\mathsf{Type}}\ \beta^{\mathsf{Type}}. \lambda \langle p, r \rangle^{\langle \alpha, \beta \to C' \rangle}\ k^{\langle \alpha, \beta \to C' \rangle \to C'}. k\ \langle p, \lambda x^{\beta}. r\ x\ k'\ h' \rangle)\ y\ h' \\
C' &= \llbracket E' \rrbracket \gamma \to \gamma
\end{aligned}$$

Performing the typed CPS translation (with forwarding) followed by type erasure yields the same result as performing the untyped translation of §4.2.1.

Our shapes are similar to the type schemes in Berthomieu and Sagazan's tagged types [3]. They can also be encoded using something similar to Remy's generalised record algebras [28]. We defer a full investigation of our extended row type system and typed variants of the uncurried CPS translations to future work.

Infinite Runs in Abstract Completion*

Nao Hirokawa[1], Aart Middeldorp[2], Christian Sternagel[3], and Sarah Winkler[4]

1 **School of Information Science, JAIST, Nomi, Japan**
`hirokawa@jaist.ac.jp`
2 **Department of Computer Science, University of Innsbruck, Innsbruck, Austria**
`aart.middeldorp@uibk.ac.at`
3 **Department of Computer Science, University of Innsbruck, Innsbruck, Austria**
`christian.sternagel@uibk.ac.at`
4 **Department of Computer Science, University of Innsbruck, Innsbruck, Austria**
`sarah.winkler@uibk.ac.at`

Abstract

Completion is one of the first and most studied techniques in term rewriting and fundamental to automated reasoning with equalities. In an earlier paper we presented a new and formalized correctness proof of abstract completion for finite runs. In this paper we extend our analysis and our formalization to infinite runs, resulting in a new proof that fair infinite runs produce complete presentations of the initial equations. We further consider ordered completion—an important extension of completion that aims to produce ground-complete presentations of the initial equations. Moreover, we revisit and extend results of Métivier concerning canonicity of rewrite systems. All proofs presented in the paper have been formalized in Isabelle/HOL.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases term rewriting, abstract completion, ordered completion, canonicity, Isabelle/HOL

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.19

1 Introduction

Reasoning with equalities is pervasive in computer science and mathematics, and has consequently been one of the main research areas of automated deduction. Indeed completion as introduced by Knuth and Bendix [12] has evolved into a fundamental technique whose ideas appear throughout automated deduction whenever equalities are present. Many variants of the original calculus have since been proposed.

On a given set of input equalities, Knuth-Bendix completion can behave in three different ways: it may (1) succeed to compute a complete system in finitely many steps, (2) fail due to unorientable equalities, or (3) continuously compute approximations of a complete system without ever terminating.

As a remedy to problem (2), ordered completion was developed by Bachmair, Dershowitz, and Plaisted [5]. Ordered completion never fails, though the price to be paid is that the resulting system is in general only complete on ground terms. This is actually sufficient for many applications in theorem proving. However, it is still possible that a ground-complete system is only produced in the limit.

* This work is supported by JSPS KAKENHI Grant Number 17K00011, JSPS Core to Core Program, and FWF (Austrian Science Fund) projects P27502, P27528, and T789.

▶ **Example 1.** Consider the equational system $\mathcal{E} = \{\mathsf{aba} \approx \mathsf{bab}\}$ of the three-strand positive braid monoid. Kapur and Narendran [10] proved that $\mathcal{E}$ admits no *finite* complete presentation. However, taking the Knuth-Bendix order [12] with a and b of weight 1 and $\mathsf{a} > \mathsf{b}$ in the precedence, completion produces in the limit the following infinite complete presentation of $\mathcal{E}$

$$\{\mathsf{aba} \to \mathsf{bab}\} \cup \{\mathsf{ab}^n\mathsf{ab} \to \mathsf{babba}^{n-1} \mid n \geqslant 2\}$$

which can be used to decide the validity problem for $\mathcal{E}$.[1]

Bachmair, Dershowitz, and Hsiang [4] recast completion procedures in inference systems. This style of presentation, *abstract completion*, has become the standard to describe completion procedures and *proof orders* the accompanying way to establish correctness [4, 5, 2]. In earlier work [8] we presented a new correctness proof for Knuth-Bendix completion which does not rely on proof orders and was entirely formalized in Isabelle/HOL [14] as part of the formal IsaFoR[2] library. However, our results were limited to finite runs.

In the present paper we adapt our proof techniques to show correctness of both Knuth-Bendix and ordered completion for potentially infinite runs. Though we emphasize the infinite case, all results are valid for finite runs, too, and thus also apply to completion procedures in practice. We believe that our proof techniques are better suited to formalization as they are more *local* than the original approach in multiple respects. We will point out differences in Sections 3 and 4.

Completion procedures raise the question whether their result is uniquely determined. Métivier [13] showed that indeed canonical rewrite systems are unique up to renaming, once a reduction order is fixed. We provide a new proof for a generalization of this result.

Contribution. We present new and comparatively short correctness proofs of Knuth-Bendix completion and ordered completion, as well as some results about canonicity, one of which generalizes the uniqueness result for complete systems due to Métivier [13]. All the proofs that are presented in the following, have been formalized as part of IsaFoR (version 2.30). With the exception of Knuth-Bendix completion for finite runs, to the best of our knowledge none of these techniques has been formalized in a proof assistant before. Also note that in the PDF version of this paper all corollary/lemma/theorem statements are active hyperlinks to HTML versions of our formalized proofs.

Overview. The remainder of this paper is organized as follows. In Section 2 we present required preliminaries followed by some mostly known results. Then, in Section 3, we recall the inference rules for (abstract) Knuth-Bendix completion and present our new correctness proof for infinite runs. Afterwards, in Section 4, we deal with ordered completion. Finally, we give some canonicity results that are related to normalization equivalence in Section 5, before we conclude in Section 6 with related and future work.

2 Preliminaries

We assume familiarity with the basic notions of abstract rewrite systems (ARSs), term rewrite systems (TRSs), and completion [2, 1], but shortly recapitulate terminology and

[1] Burckel [6] constructed a complete rewrite system consisting of four rules with an additional symbol, which is no longer a presentation of $\mathcal{E}$ but can be also used to decide the validity problem for $\mathcal{E}$.

[2] The *Isabelle Formalization of Rewriting*: `http://cl-informatik.uibk.ac.at/isafor/`

notation that we use in the remainder. For an arbitrary binary relation $\to_\alpha$, we write ${}_\alpha\!\leftarrow$, $\underset{\alpha}{\leftrightarrow}$, $\to_\alpha^=$, $\to_\alpha^+$, and $\to_\alpha^*$ to denote its *inverse*, its *symmetric closure*, its *reflexive closure*, its *transitive closure*, and its *reflexive transitive closure*, respectively. We further use $\downarrow_\alpha$ as abbreviation for the relation $\to_\alpha^* \cdot {}_\alpha^*\!\leftarrow$, where from here on $\cdot$ denotes relation composition. If $a \to_\alpha b$ for no b then we say that a is a *normal form* of $\to_\alpha$ and write $a \in \mathsf{NF}(\to_\alpha)$. By $a \to_\alpha^! b$ we abbreviate $a \to_\alpha^* b \wedge b \in \mathsf{NF}(\to_\alpha)$. Such an element b is called a normal form of a. Given two binary relations $\to_\alpha$ and $\to_\beta$, we use $\to_\alpha \,/\, \to_\beta$ as shorthand for $\to_\beta^* \cdot \to_\alpha \cdot \to_\beta^*$. A *renaming* is a bijective variable substitution from $\mathcal{V}$ to $\mathcal{V}$. A term s is a *variant* of a term t if $s = t\sigma$ for some renaming σ. If $\ell \to r$ is a rewrite rule and σ is a renaming then the rewrite rule $\ell\sigma \to r\sigma$ is a variant of $\ell \to r$. A TRS is said to be *variant-free* if it does not contain rewrite rules that are variants of each other. Given terms s and t, we write $s \doteq t$ if $s\sigma = t$ and $s = t\tau$ for some substitutions σ and τ. We say that s *encompasses* t, written $s \trianglerighteq t$, whenever $s = C[t\sigma]$ for some context C and substitution σ. *Proper encompassment* is defined by $\triangleright = \trianglerighteq \setminus \trianglelefteq$ and known to be well-founded. Two variable-disjoint variants $\ell_1 \to r_1$ and $\ell_2 \to r_2$ of rules in $\mathcal{R}$ such that $\ell_1\mu = \ell_2|_p\mu$ with $p \in \mathcal{P}\mathsf{os}_\mathcal{F}(\ell_2)$ and most general unifier (mgu) μ, constitute an *overlap*. An overlap that does not result from overlapping two variants of the same rule at the root, gives rise to a *critical pair* $\ell_2[r_1]_p\mu \approx r_2\mu$. A critical pair is called *prime* if all proper subterms of $\ell_1\mu$ are $\mathcal{R}$-normal forms. The set of (prime) critical pairs of a TRS $\mathcal{R}$ is denoted by $(\mathsf{PCP}(\mathcal{R}))$ $\mathsf{CP}(\mathcal{R})$. For a well-founded order $>$, we write $>_{\mathrm{mul}}$ to denote its *multiset extension* and $>_{\mathrm{lex}}$ to denote its *lexicographic extension* as defined by Baader and Nipkow [1].

We make use of the following result due to Bachmair and Dershowitz [3]. Here *quasi-commutation* of R over S means that the inclusion $S \cdot R \subseteq R \cdot (R \cup S)^*$ holds.

▶ **Lemma 2.** *Let R and S be binary relations.*
1. *If R quasi-commutes over S and R is well-founded then $R \,/\, S$ is well-founded.*
2. *If $R \,/\, S$ and S are well-founded then $R \cup S$ is well-founded.*

▶ **Lemma 3.** *If R is a well-founded rewrite relation then $(R \cup \triangleright) \,/\, \trianglerighteq$ is well-founded.*

Proof. First we show the inclusion $\trianglerighteq \cdot R \subseteq R \cdot \trianglerighteq$. Suppose $s \trianglerighteq t\ R\ u$. So $s = C[t\sigma]$ for some context C and substitution σ. Because R is closed under contexts and substitutions, $s\ R\ C[u\sigma]$. Moreover, $C[u\sigma] \trianglerighteq u$. This establishes the inclusion, and we conclude that R (quasi-)commutes over $\trianglerighteq$. Because R is well-founded, it follows from Lemma 2(1) that the relation $R \,/\, \trianglerighteq$ is well-founded too. Then $R \,/\, \triangleright$ is well-founded since it is contained in $R \,/\, \trianglerighteq$. As $\triangleright$ is well-founded, it follows from Lemma 2(2) that $R \cup \triangleright$ is well-founded. We have $\trianglerighteq \cdot \triangleright \subseteq \triangleright$ and thus $R \cup \triangleright$ quasi-commutes over $\trianglerighteq$. Another application of Lemma 2(1) yields the well-foundedness of $(R \cup \triangleright) \,/\, \trianglerighteq$. ◀

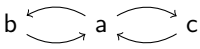
We use the following simple confluence criterion for ARSs. Let $\mathcal{A} = \langle A, \to \rangle$ be an ARS equipped with a well-founded relation $>$ on A, and let $b \overset{a}{\to} c$ if and only if $b \to c$ and $a = b$. We say that $\mathcal{A}$ is *source decreasing* if the inclusion

$$\leftarrow a \to \ \subseteq \overset{\vee a}{\longleftrightarrow}{}^*$$

holds for all $a \in A$. Here $\overset{\vee a}{\longleftrightarrow}{}^*$ denotes a conversion in which all steps are labeled with an element smaller than a. Source decreasingness is the specialization of peak-decreasingness [8] to source labeling [17, Example 6].

▶ **Lemma 4.** *Every source decreasing ARS is confluent.*

Source-decreasingness is closely related to the *connectedness-below* criterion of Winkler and Buchberger [18]. Unlike the latter, it does not entail termination. For instance, for $\mathsf{a} > \mathsf{b}$ and $\mathsf{a} > \mathsf{c}$ the non-terminating ARS



is source decreasing but the connectedness-below criterion does not apply.

The following definition and corresponding lemma [8] are key to the correctness results for both Knuth-Bendix completion and ordered completion.

▶ **Definition 5.** Given a TRS $\mathcal{R}$ and terms s, t, and u, we write $t \mathrel{\triangledown_s} u$ if $s \to_{\mathcal{R}}^{+} t$, $s \to_{\mathcal{R}}^{+} u$, and $t \downarrow_{\mathcal{R}} u$ or $t \leftrightarrow_{\mathsf{PCP}(\mathcal{R})} u$.

▶ **Lemma 6.** *Let $\mathcal{R}$ be a TRS. If $t \mathrel{{}_{\mathcal{R}}\!\leftarrow} s \to_{\mathcal{R}} u$ then $t \mathrel{\triangledown_s^2} u$.*

3 Knuth-Bendix Completion

The original completion procedure by Knuth and Bendix [12] was presented as a concrete algorithm. Later on, Bachmair, Dershowitz, and Hsiang [4] presented an inference system for completion and showed that all *fair* implementations thereof (in particular the original procedure) are correct. Abstracting from a concrete strategy, their approach thus has the advantage to cover a variety of implementations. Below, we recall the inference system, which constitutes the basis of the results presented in this section.

▶ **Definition 7.** The inference system KB of abstract (Knuth-Bendix) completion operates on pairs $(\mathcal{E}, \mathcal{R})$ of equations $\mathcal{E}$ and rules $\mathcal{R}$ over a common signature $\mathcal{F}$. It consists of the following inference rules:

$$\textsf{deduce}\quad \frac{\mathcal{E}, \mathcal{R}}{\mathcal{E} \cup \{s \approx t\}, \mathcal{R}} \text{ if } s \mathrel{{}_{\mathcal{R}}\!\leftarrow} \cdot \to_{\mathcal{R}} t \qquad \textsf{compose}\quad \frac{\mathcal{E}, \mathcal{R} \uplus \{s \to t\}}{\mathcal{E}, \mathcal{R} \cup \{s \to u\}} \text{ if } t \to_{\mathcal{R}} u$$

$$\textsf{orient}\quad \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{s \to t\}} \text{ if } s > t \qquad \textsf{simplify}\quad \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E} \cup \{u \approx t\}, \mathcal{R}} \text{ if } s \to_{\mathcal{R}} u$$

$$\frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{t \to s\}} \text{ if } t > s \qquad \frac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E} \cup \{s \approx u\}, \mathcal{R}} \text{ if } t \to_{\mathcal{R}} u$$

$$\textsf{delete}\quad \frac{\mathcal{E} \uplus \{s \approx s\}, \mathcal{R}}{\mathcal{E}, \mathcal{R}} \qquad \textsf{collapse}\quad \frac{\mathcal{E}, \mathcal{R} \uplus \{t \to s\}}{\mathcal{E} \cup \{u \approx s\}, \mathcal{R}} \text{ if } t \xrightarrow{\rhd}_{\mathcal{R}} u$$

Here $>$ is a fixed reduction order on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and the relation $t \xrightarrow[\mathcal{R}]{\rhd} u$ is defined as $t \xrightarrow[\ell \to r]{} u$ for some $\ell \to r \in \mathcal{R}$ such that $t \rhd \ell$.

Sternagel and Thiemann [15] showed that the strict encompassment condition in the collapse inference rule is not necessary for finite runs. For infinite runs however, it is indispensable: when omitted, the result need not be confluent [1, Example 7.2.9].

We write $(\mathcal{E}, \mathcal{R}) \vdash (\mathcal{E}', \mathcal{R}')$ if $(\mathcal{E}', \mathcal{R}')$ can be obtained from $(\mathcal{E}, \mathcal{R})$ by applying one of the inference rules of Definition 7. A *run* of Knuth-Bendix completion is an infinite sequence of the form

$$\Gamma\colon (\mathcal{E}_0, \mathcal{R}_0) \vdash (\mathcal{E}_1, \mathcal{R}_1) \vdash (\mathcal{E}_2, \mathcal{R}_2) \vdash \cdots$$

with $\mathcal{R}_0 = \varnothing$. We define

$$\mathcal{E}_\infty = \bigcup_{i \geqslant 0} \mathcal{E}_i\,, \qquad \mathcal{R}_\infty = \bigcup_{i \geqslant 0} \mathcal{R}_i\,, \qquad \mathcal{E}_\omega = \bigcup_{i \geqslant 0} \bigcap_{j \geqslant i} \mathcal{E}_j\,, \qquad \mathcal{R}_\omega = \bigcup_{i \geqslant 0} \bigcap_{j \geqslant i} \mathcal{R}_j\,.$$

Equations in $\mathcal{E}_\omega$ and rules in $\mathcal{R}_\omega$ are called *persistent*. Note that any finite run with final state $(\mathcal{E}_n, \mathcal{R}_n)$ can be extended to an infinite run (e.g., by deduce steps followed by delete steps) such that $(\mathcal{E}_\omega, \mathcal{R}_\omega) = (\mathcal{E}_n, \mathcal{R}_n)$. Hence the following results, even though stated for infinite derivations, also capture the finite deductions of practical completion tools. The run Γ is called *non-failing* if $\mathcal{E}_\omega = \varnothing$, and *fair* if the inclusion

$$\mathsf{PCP}(\mathcal{R}_\omega) \subseteq \bigcup_{i \geqslant 0} \overset{}{\underset{\mathcal{E}_i}{\leftrightarrow}}$$

holds. Bachmair et al. [4] proved that for every non-failing fair run, the TRS $\mathcal{R}_\omega$ constitutes a complete presentation of $\mathcal{E}_0$. The remainder of this section is dedicated to establish the same result, but on a different route without encountering proof orders.

We start by showing a few properties of inference steps of completion. In the following proofs, they allow us to keep track of how equations and rules are modified during the completion process without caring about which inference rule was actually applied.

▶ **Lemma 8.** *Suppose* $(\mathcal{E}, \mathcal{R}) \vdash (\mathcal{E}', \mathcal{R}')$. *Then the following inclusions hold:*

1. $\mathcal{E}' \cup \mathcal{R}' \subseteq \leftrightarrow^*_{\mathcal{E} \cup \mathcal{R}}$,
2. $\mathcal{E} \setminus \mathcal{E}' \subseteq (\to_{\mathcal{R}'} \cdot \mathcal{E}') \cup (\mathcal{E}' \cdot {}_{\mathcal{R}'}\!\leftarrow) \cup \mathcal{R}' \cup \mathcal{R}'^{-1} \cup =$,
3. $\mathcal{R} \setminus \mathcal{R}' \subseteq (\overset{\rhd}{\longrightarrow}_{\mathcal{R}'} \cdot \mathcal{E}') \cup (\mathcal{R}' \cdot {}_{\mathcal{R}'}\!\leftarrow)$.

Together these properties reveal that inference steps do not change the conversion relation:

▶ **Corollary 9.** *If* $(\mathcal{E}, \mathcal{R}) \vdash (\mathcal{E}', \mathcal{R}')$ *then the relations* $\overset{*}{\underset{\mathcal{E} \cup \mathcal{R}}{\longleftrightarrow}}$ *and* $\overset{*}{\underset{\mathcal{E}' \cup \mathcal{R}'}{\longleftrightarrow}}$ *coincide.*

Below, we consider the infinite non-failing run $\Gamma\colon (\mathcal{E}_0, \mathcal{R}_0) \vdash (\mathcal{E}_1, \mathcal{R}_1) \vdash (\mathcal{E}_2, \mathcal{R}_2) \vdash \cdots$. First we show that all rewrite rules are compatible with the reduction order $>$.

▶ **Lemma 10.** *The inclusions* $\mathcal{R}_\omega \subseteq \mathcal{R}_\infty \subseteq >$ *hold.*

Next, we verify that every equality in $\mathcal{E}_i$ can be turned into a valley in $\mathcal{R}_\infty$. Note that in contrast to the proof order approach [4] and to our previous correctness proof for finite runs [8] we reason separately about equations and rules. This more local rationale simplifies the analysis as we can use different well-founded induction arguments for the two cases, rather than synthesizing an order that covers both.

▶ **Lemma 11.** *The inclusion* $\mathcal{E}_i \subseteq \downarrow_{\mathcal{R}_\infty}$ *holds for all* $i \geqslant 0$.

Proof. Let $s \approx t \in \mathcal{E}_i$. By induction on $\{s, t\}$ with respect to $>_{\mathrm{mul}}$ we show $s \downarrow_{\mathcal{R}_\infty} t$. Because $\mathcal{E}_\omega = \varnothing$, $s \approx t \in \mathcal{E}_{j-1} \setminus \mathcal{E}_j$ for some $j > i$. Following Lemma 8(2), we distinguish three cases.

- If $s \approx t \in \mathcal{R}_j \cup \mathcal{R}_j^{-1} \cup =$ then the claim trivially holds.
- If $s \to_{\mathcal{R}_j} u$ and $u \approx t \in \mathcal{E}_j$ for some term u then $\{s, t\} >_{\mathrm{mul}} \{u, t\}$ and thus $u \downarrow_{\mathcal{R}_\infty} t$ by the induction hypothesis. Hence also $s \downarrow_{\mathcal{R}_\infty} t$.
- Similarly, if $s \approx u \in \mathcal{E}_j$ and $u \;{}_{\mathcal{R}_j}\!\leftarrow t$ for some term u then $\{s, t\} >_{\mathrm{mul}} \{s, u\}$ and we obtain $s \downarrow_{\mathcal{R}_\infty} t$ as in the preceding case. ◀

▶ **Corollary 12.** *The inclusion* $\underset{\mathcal{E}_i}{\longrightarrow} \subseteq \overset{*}{\underset{\mathcal{R}_\infty}{\longleftrightarrow}}$ *holds for all* $i \geqslant 0$.

In order to show confluence of $\mathcal{R}_\omega$ we use source labeling to label steps in $\mathcal{R}_\infty$. The next lemma allows us to transform every non-persistent rule $\ell \to r$ into an $\mathcal{R}_\omega$-conversion below ℓ. In the proof we employ the following extension of the reduction order $>$ used in completion.

▶ **Definition 13.** We define $\succ = ((> \cup \rhd) \,/\, \trianglerighteq)^+$.

According to Lemma 3, $\succ$ is a well-founded order.

▶ **Lemma 14.** *The inclusion* $\xrightarrow[\mathcal{R}_\infty]{s} \subseteq \overset{\mathbb{V}\, s}{\underset{\mathcal{R}_\omega}{\longleftrightarrow}}{}^*$ *holds for all terms* s.

Proof. Let $s \xrightarrow{s}_{\mathcal{R}_\infty} t$ by employing the rewrite rule $\ell \to r$. We prove $s \overset{\mathbb{V}\, s}{\longleftrightarrow}{}^*_{\mathcal{R}_\omega} t$ by induction on (ℓ, r) with respect to $\succ_{\mathrm{lex}}$. If $\ell \to r \in \mathcal{R}_\omega$ then the claim trivially holds. Otherwise, $\ell \to r \in \mathcal{R}_{i-1} \setminus \mathcal{R}_i$ for some $i > 0$. Using Lemma 8(3), we distinguish two cases.

- Suppose $\ell \xrightarrow{\rhd}_{\ell' \to r'} u$ and $u \approx r \in \mathcal{E}_i$ for some term u and rule $\ell' \to r' \in \mathcal{R}_i$. We obtain $\ell \xrightarrow{\rhd}_{\ell' \to r'} u \downarrow_{\mathcal{R}_\infty} r$ from Lemma 11. We have $\ell \rhd \ell'$ and both $\ell > u$ and $\ell > r$. It follows that all rewrite rules $\ell'' \to r''$ employed in $\ell \xrightarrow{\rhd} u \downarrow_{\mathcal{R}_\infty} r$ satisfy $(\ell, r) \succ_{\mathrm{lex}} (\ell'', r'')$. Moreover, all steps in $\ell \downarrow_{\mathcal{R}_\infty} r$ are labeled with a term $\leqslant \ell$. Hence we obtain $\ell \overset{\mathbb{V}\, \ell}{\longleftrightarrow}{}^*_{\mathcal{R}_\omega} r$ from the induction hypothesis.
- Suppose $\ell \to u \in \mathcal{R}_i$ and $u \leftarrow_{\ell' \to r'} r$ for some term u and rewrite rule $\ell' \to r' \in \mathcal{R}_i$. We have $(\ell, r) \succ_{\mathrm{lex}} (\ell, u)$ and $(\ell, r) \succ_{\mathrm{lex}} (\ell', r')$. Moreover, both steps are labeled with a term $\leqslant \ell$ and thus we obtain $\ell \overset{\mathbb{V}\, \ell}{\longleftrightarrow}{}^*_{\mathcal{R}_\omega} r$ from the induction hypothesis.

So in both cases we have $\ell \overset{\mathbb{V}\, \ell}{\longleftrightarrow}{}^*_{\mathcal{R}_\omega} r$ and thus also $s \overset{\mathbb{V}\, s}{\longleftrightarrow}{}^*_{\mathcal{R}_\omega} t$. ◀

▶ **Corollary 15.** *The relations* $\overset{*}{\underset{\mathcal{R}_\infty}{\longleftrightarrow}}$ *and* $\overset{*}{\underset{\mathcal{R}_\omega}{\longleftrightarrow}}$ *coincide.*

We arrive at the main theorem of this section. Note that Bachmair's correctness proof [2] uses induction with respect to a well-founded order on conversions to directly show that any conversion of $\mathcal{E}_\infty \cup \mathcal{R}_\infty$ can be transformed into a joining sequence of $\mathcal{R}_\omega$. In contrast, we prove confluence via source decreasingness. This allows us to concentrate on *local* peaks.

▶ **Theorem 16.** *If* Γ *is fair then* $\mathcal{R}_\omega$ *is a complete presentation of* $\mathcal{E}_0$.

Proof. We have $\mathcal{E}_\omega = \varnothing$ because Γ is non-failing. The TRS $\mathcal{R}_\omega$ is terminating by Lemma 10. We show source decreasingness of labeled $\mathcal{R}_\omega$ reduction with respect to the reduction order $>$. So let $t \;{}_{\mathcal{R}_\omega}\!\overset{s}{\leftarrow} s \xrightarrow{s}_{\mathcal{R}_\omega} u$. From Lemma 6 we obtain $t \mathrel{\triangledown_s^2} u$. Let $v \mathrel{\triangledown_s} w$ appear in this sequence (so $t = v$ or $w = u$). We have $s > v$, $s > w$, and

$$(v, w) \in \downarrow_{\mathcal{R}_\omega} \cup \bigcup_{i \geqslant 0} \leftrightarrow_{\mathcal{E}_i}$$

by the definition of $\triangledown_s$ and fairness of Γ.

- If $v \downarrow_{\mathcal{R}_\omega} w$ then $v \xrightarrow{\mathbb{V}\, v}{}^*_{\mathcal{R}_\omega} \cdot \;{}^*_{\mathcal{R}_\omega}\!\overset{\mathbb{V}\, w}{\leftarrow} w$ and thus $v \overset{\vee s}{\longleftrightarrow}{}^*_{\mathcal{R}_\omega} w$.
- If $v \leftrightarrow_{\mathcal{E}_i} w$ for some $i \geqslant 0$ then $v \downarrow_{\mathcal{R}_\infty} w$ by Lemma 11. We obtain $v \overset{\vee s}{\longleftrightarrow}{}^*_{\mathcal{R}_\infty} w$ as in the previous case and thus $v \overset{\vee s}{\longleftrightarrow}{}^*_{\mathcal{R}_\omega} w$ by Lemma 14.

Hence $t \overset{\vee s}{\longleftrightarrow}{}^*_{\mathcal{R}_\omega} u$. Confluence of $\mathcal{R}_\omega$ now follows from Lemma 4. It remains to show $\leftrightarrow^*_{\mathcal{E}_0} = \leftrightarrow^*_{\mathcal{R}_\omega}$. Using Corollary 9 we obtain $\to_{\mathcal{E}_i \cup \mathcal{R}_i} \subseteq \leftrightarrow^*_{\mathcal{E}_0}$ by a straightforward induction on i. This in turn yields $\leftrightarrow^*_{\mathcal{E}_0} = \leftrightarrow^*_{\mathcal{E}_\infty \cup \mathcal{R}_\infty}$. From Corollary 12 we infer $\leftrightarrow^*_{\mathcal{E}_\infty \cup \mathcal{R}_\infty} = \leftrightarrow^*_{\mathcal{R}_\infty}$ and we conclude by an appeal to Corollary 15. ◀

▶ **Example 17.** Consider the equational system $\mathcal{E}$ and the KBO $>$ from Example 1. Let $\mathcal{P}_n$ denote the TRS $\{\mathsf{ab}^{i+1}\mathsf{ab} \to \mathsf{babba}^i \mid 1 \leqslant i \leqslant n\}$. One possible infinite completion run is the following:

$$\begin{aligned}
(\mathcal{E}, \varnothing) &\vdash^{\mathsf{orient}} (\varnothing, \{\mathsf{aba} \to \mathsf{bab}\}) && \vdash^{\mathsf{deduce}} (\{\mathsf{abbab} \approx \mathsf{babba}\}, \{\mathsf{aba} \to \mathsf{bab}\}) \\
&\vdash^{\mathsf{orient}} (\varnothing, \{\mathsf{aba} \to \mathsf{bab}\} \cup \mathcal{P}_1) && \vdash^{\mathsf{deduce}} (\{\mathsf{abbbab} \approx \mathsf{babbaa}\}, \{\mathsf{aba} \to \mathsf{bab}\} \cup \mathcal{P}_1) \\
&\vdash^{\mathsf{orient}} (\varnothing, \{\mathsf{aba} \to \mathsf{bab}\} \cup \mathcal{P}_2) && \vdash \cdots
\end{aligned}$$

If this run is continued in a fair way we subsequently construct the TRSs $\mathcal{P}_n$ and can in the limit obtain the result $\mathcal{R}_\omega = \{\mathsf{aba} \to \mathsf{bab}\} \cup \{\mathsf{ab}^{i+1}\mathsf{ab} \to \mathsf{babba}^i \mid 1 \leqslant i\}$, which is complete according to Theorem 16.

4 Ordered Completion

In this section $>$ is a fixed ground-total reduction order, i.e., for all ground terms $s, t \in \mathcal{T}(\mathcal{F})$ either $s > t$, $t > s$, or $s = t$ holds. Given a binary relation R, we write $R^\pm$ for the symmetric closure $R \cup R^{-1}$. For a set $\mathcal{E}$ of equations, an ordered rewrite step is a rewrite step using a rule from $\mathcal{E}^>$, which is the set of rewrite rules $\ell\sigma \to r\sigma$ such that $\ell \approx r \in \mathcal{E}^\pm$ and $\ell\sigma > r\sigma$.

An *extended overlap* is given by two variable-disjoint variants $\ell_1 \approx r_1$ and $\ell_2 \approx r_2$ of equations in $\mathcal{E}^\pm$ such that $\ell_1\mu = \ell_2|_p\mu$ with $p \in \mathcal{P}\mathsf{os}_\mathcal{F}(\ell_2)$ and mgu μ. An extended overlap which satisfies $r_1\mu \not> \ell_1\mu$ and $r_2\mu \not> \ell_2\mu$ gives rise to the *extended critical pair* $\ell_2[r_1]_p\mu \approx r_2\mu$. An extended critical pair is called *prime* if all proper subterms of $\ell_1\mu$ are $\mathcal{E}^>$-normal forms. The set of extended prime critical pairs among equations in $\mathcal{E}$ is denoted $\mathsf{PCP}_>(\mathcal{E})$.

The following inference rules for ordered completion are due to Bachmair, Dershowitz, and Plaisted [5]. In order to simplify the notation, we abbreviate $\mathcal{R} \cup \mathcal{E}^>$ to $\mathcal{S}$, and use the following shorthands. We write $t \xrightarrow{\rhd}_{\mathcal{E}^>} u$ if there exist an equation $\ell \approx r \in \mathcal{E}^\pm$, a context C, and a substitution σ such that $t = C[\ell\sigma]$, $u = C[r\sigma]$, $\ell\sigma > r\sigma$, and $t \rhd \ell$. The union of $\to_\mathcal{R}$ and $\xrightarrow{\rhd}_{\mathcal{E}^>}$ is denoted by $\xrightarrow{\rhd_1}_\mathcal{S}$ and we write $\xrightarrow{\rhd_2}_\mathcal{S}$ for the union of $\xrightarrow{\rhd}_\mathcal{R}$ and $\xrightarrow{\rhd}_{\mathcal{E}^>}$.

▶ **Definition 18.** The inference system oKB of ordered completion operates on pairs $(\mathcal{E}, \mathcal{R})$ of equations $\mathcal{E}$ and rules $\mathcal{R}$ over a common signature $\mathcal{F}$. It consists of the following inference rules:

deduce	$\dfrac{\mathcal{E}, \mathcal{R}}{\mathcal{E} \cup \{s \approx t\}, \mathcal{R}}$	if $s \xleftarrow[\mathcal{R}\cup\mathcal{E}]{} \cdot \xrightarrow[\mathcal{R}\cup\mathcal{E}]{} t$	compose	$\dfrac{\mathcal{E}, \mathcal{R} \uplus \{s \to t\}}{\mathcal{E}, \mathcal{R} \cup \{s \to u\}}$	if $t \to_\mathcal{S} u$
orient	$\dfrac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{s \to t\}}$	if $s > t$	simplify	$\dfrac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E} \cup \{u \approx t\}, \mathcal{R}}$	if $s \xrightarrow{\rhd_1}_\mathcal{S} u$
	$\dfrac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E}, \mathcal{R} \cup \{t \to s\}}$	if $t > s$		$\dfrac{\mathcal{E} \uplus \{s \approx t\}, \mathcal{R}}{\mathcal{E} \cup \{s \approx u\}, \mathcal{R}}$	if $t \xrightarrow{\rhd_1}_\mathcal{S} u$
delete	$\dfrac{\mathcal{E} \uplus \{s \approx s\}, \mathcal{R}}{\mathcal{E}, \mathcal{R}}$		collapse	$\dfrac{\mathcal{E}, \mathcal{R} \uplus \{t \to s\}}{\mathcal{E} \cup \{u \approx s\}, \mathcal{R}}$	if $t \xrightarrow{\rhd_2}_\mathcal{S} u$

The deduce rule may be applied to any peak, though in practice it is typically limited to the addition of extended critical pairs. We write $(\mathcal{E}, \mathcal{R}) \vdash_\mathsf{o} (\mathcal{E}', \mathcal{R}')$ if $(\mathcal{E}', \mathcal{R}')$ can be reached from $(\mathcal{E}, \mathcal{R})$ by employing one of the inference rules of Definition 18. We start by stating the equivalents of Lemma 8 and Corollary 9 for ordered completion.

▶ **Lemma 19.** *Suppose $(\mathcal{E}, \mathcal{R}) \vdash_\mathsf{o} (\mathcal{E}', \mathcal{R}')$. Then the following inclusions hold:*

1. $\mathcal{E}' \cup \mathcal{R}' \subseteq \xleftrightarrow[\mathcal{E}\cup\mathcal{R}]{*}$,
2. $\mathcal{E} \setminus \mathcal{E}' \subseteq (\xrightarrow[\mathcal{S}']{\rhd_1} \cdot \mathcal{E}'^\pm)^\pm \cup \mathcal{R}'^\pm \cup =$,
3. $\mathcal{R} \setminus \mathcal{R}' \subseteq (\xrightarrow[\mathcal{S}']{\rhd_2} \cdot \mathcal{E}') \cup (\mathcal{R}' \cdot \xleftarrow[\mathcal{S}']{})$.

▶ **Corollary 20.** *If $(\mathcal{E}, \mathcal{R}) \vdash_\mathsf{o} (\mathcal{E}', \mathcal{R}')$ then the relations $\xleftrightarrow[\mathcal{E}\cup\mathcal{R}]{*}$ and $\xleftrightarrow[\mathcal{E}'\cup\mathcal{R}']{*}$ coincide.*

Below, we consider the infinite run $\Gamma\colon (\mathcal{E}_0, \mathcal{R}_0) \vdash_\mathsf{o} (\mathcal{E}_1, \mathcal{R}_1) \vdash_\mathsf{o} (\mathcal{E}_2, \mathcal{R}_2) \vdash_\mathsf{o} \cdots$.

▶ **Lemma 21.** *The inclusions $\mathcal{R}_\omega \subseteq \mathcal{R}_\infty \subseteq >$ and $\mathcal{E}_\omega \subseteq \mathcal{E}_\infty$ hold.*

Unlike for Knuth-Bendix completion we do not assume Γ to be non-failing, and in general $\mathcal{E}_i \subseteq \downarrow_{\mathcal{R}_\infty}$ does not hold. So we take a different route. Given a rewrite relation $\to$ and a set S of terms, we write $t \xrightarrow{S} u$ if $t \to u$, $s \geqslant t$, and $s' \geqslant u$ for some terms $s, s' \in S$. Since both $\to$ and $\geqslant$ are closed under contexts and substitutions, we have $C[t\sigma] \xrightarrow{S'} C[u\sigma]$ whenever $t \xrightarrow{S} u$ and $S' = \{C[s\sigma] \mid s \in S\}$, for all contexts C and substitutions σ. We use this relation to show that any equation step below a term set S eventually turns into a conversion over $\mathcal{R}_\infty \cup \mathcal{E}_\omega$ that is still below S. Note that just like in Section 3 we avoid the use of a synthesized termination argument by handling equations and rules separately.

▶ **Lemma 22.** *The inclusion $\xrightarrow[\mathcal{E}_\infty]{S} \subseteq \xleftrightarrow[\mathcal{R}_\infty \cup \mathcal{E}_\omega]{S}{}^*$ holds for all sets S of terms.*

Proof. Let $t \approx u \in \mathcal{E}_\infty$. We prove

$$\xrightarrow[t \approx u]{S} \subseteq \xleftrightarrow[\mathcal{R}_\infty \cup \mathcal{E}_\omega]{S}{}^*$$

by induction on $\{t, u\}$ with respect to the well-founded order $\succ_{\mathrm{mul}}$. If $t \approx u \in \mathcal{E}_\omega^{\pm}$ then the claim follows trivially. Otherwise, $t \approx u \in (\mathcal{E}_{i-1} \setminus \mathcal{E}_i)^{\pm}$ for some $i > 0$. Using Lemma 19(2), we distinguish two subcases.

- Suppose $t \approx u \in (\xrightarrow{\rhd_1}_{\mathcal{S}_i} \cdot \mathcal{E}_i^{\pm})^{\pm}$. There exist a term t' and an equation $v' \approx u' \in \mathcal{E}_i^{\pm}$ such that $\{t, u\} = \{t', u'\}$ and $t' \xrightarrow{\rhd_1}_{\mathcal{S}_i} v'$. It is sufficient to show

  $$t' \xleftrightarrow[\mathcal{R}_\infty \cup \mathcal{E}_\omega]{\{t', u'\}}{}^* v' \quad \text{and} \quad v' \xleftrightarrow[\mathcal{R}_\infty \cup \mathcal{E}_\omega]{\{t', u'\}}{}^* u'$$

  The second conversion follows from $t' > v'$ and the induction hypothesis for $v' \approx u' \in \mathcal{E}_i^{\pm}$, which is applicable as $\{t, u\} = \{t', u'\} \succ_{\mathrm{mul}} \{v', u'\}$. The first conversion is obtained as follows. Because of $t' \xrightarrow{\rhd_1}_{\mathcal{S}_i} v'$, we have $t' \to_{\mathcal{R}_i} v'$ or $t' \xrightarrow{\rhd}_{\mathcal{E}_i^>} v'$. If $t' \to_{\mathcal{R}_i} v'$ then this step can be labeled with $\{t', u'\}$ as $t' > v'$. Otherwise, there exist an equation $\ell \approx r \in \mathcal{E}_i^{\pm}$, a context C, and a substitution σ such that $t' = C[\ell\sigma]$, $v' = C[r\sigma]$, $\ell\sigma > r\sigma$, and $t' \rhd \ell$. We have $t' \succ \ell$ and $t' \succ r$ as $t' \trianglerighteq \ell\sigma > r\sigma \trianglerighteq r$. Therefore $\{t', u'\} \succ_{\mathrm{mul}} \{\ell, r\}$ holds, so

  $$\ell \xleftrightarrow[\mathcal{R}_\infty \cup \mathcal{E}_\omega]{\{\ell, r\}}{}^* r$$

  follows from the induction hypothesis. Closure under contexts and substitutions now yields $t \xleftrightarrow{\{t,u\}}{}^*_{\mathcal{R}_\infty \cup \mathcal{E}_\omega} u$.
- If $t \approx u \in \mathcal{R}_i^{\pm} \cup =$ then $t \xleftrightarrow[\mathcal{R}_\infty]{\{t,u\}}{}^= u$.

In both cases $t \xleftrightarrow{\{t,u\}}{}^*_{\mathcal{R}_\infty \cup \mathcal{E}_\omega} u$ holds. Since S contains upper bounds of t and u with respect to $\geqslant$, the desired inclusion follows from the closure under contexts and substitutions of $\to_{\mathcal{R}_\infty \cup \mathcal{E}_\omega}$ and $\geqslant$. ◀

Next, we show that a rewrite step that uses a rule in $\mathcal{R}_\infty$ and is below a term set S eventually turns into a conversion over persistent rules and equations that is still below S. We write Υt for the set $\{u \in \mathcal{T}(\mathcal{F}, \mathcal{V}) \mid t \succ u\}$.

▶ **Lemma 23.** *The inclusion $\xrightarrow[\mathcal{R}_\infty]{S} \subseteq \xleftrightarrow[\mathcal{R}_\omega \cup \mathcal{E}_\omega]{S}{}^*$ holds for all sets S of terms.*

Proof. Let $\ell \approx r \in \mathcal{R}_\infty$. We prove

$$\xrightarrow[\ell \to r]{S} \subseteq \xleftrightarrow[\mathcal{R}_\omega \cup \mathcal{E}_\omega]{S}{}^*$$

by induction on (ℓ, r) with respect to the well-founded order $\succ_{\mathrm{lex}}$. If $\ell \to r \in \mathcal{R}_\omega$ then the claim trivially holds. Otherwise, there is some $i > 0$ such that $\ell \to r \in \mathcal{R}_{i-1} \setminus \mathcal{R}_i$. From Lemma 22 and the induction hypothesis the inclusions

$$\xrightarrow[\mathcal{R}_\infty \cup \mathcal{E}_\infty]{T} \subseteq \xleftrightarrow[\mathcal{R}_\infty \cup \mathcal{E}_\omega]{T}{}^* \subseteq \xleftrightarrow[\mathcal{R}_\omega \cup \mathcal{E}_\omega]{T}{}^* \tag{1}$$

are obtained for every set $T \subseteq \Upsilon\ell$. Using Lemma 19, we distinguish two cases.

- Suppose $\ell \xrightarrow{\trianglerighteq_2}_{\mathcal{S}_i} u$ and $u \approx r \in \mathcal{E}_i$ for some term u. There exist an equation $\ell' \approx r' \in \mathcal{R}_\infty \cup \mathcal{E}_\infty^{\pm}$, a context C and a substitution σ such that $\ell = C[\ell'\sigma]$, $u = C[r'\sigma]$, $\ell\sigma > r\sigma$, and $\ell \trianglerighteq \ell'$. We have $\ell \succ \ell', r'$ as $\ell \trianglerighteq \ell'$ and $\ell \trianglerighteq \ell'\sigma > r'\sigma \trianglerighteq r'$ and thus $\ell' \xleftrightarrow{\{\ell'\}}_{\mathcal{R}_\infty \cup \mathcal{E}_\infty} r'$. Since $\{\ell', r'\} \subseteq \Upsilon\ell$ we obtain $\ell' \xleftrightarrow{\{\ell', r'\}}{}^*_{\mathcal{R}_\omega \cup \mathcal{E}_\omega} r'$ from (1). Therefore, $\ell \xleftrightarrow{\{\ell\}}{}^*_{\mathcal{R}_\omega \cup \mathcal{E}_\omega} u$ follows from closure under contexts and substitutions and $\ell > u$. Again from $\ell > u, r$ we obtain $u \xleftrightarrow{\Upsilon\ell}_{\mathcal{R}_\infty \cup \mathcal{E}_\infty} r$ and thus $u \xleftrightarrow{\Upsilon\ell}_{\mathcal{R}_\omega \cup \mathcal{E}_\omega} r$ follows from (1).
- Suppose $\ell \to u \in \mathcal{R}_i$ and $u \,{}_{\mathcal{S}_i}\!\leftarrow r$ for some term u. We have $r > u$ and thus $(\ell, r) \succ_{\mathrm{lex}} (\ell, u)$. Hence we can apply the induction hypothesis to $\ell \xrightarrow{\{\ell\}} u$, yielding $\ell \xleftrightarrow{\{\ell\}}{}^*_{\mathcal{R}_\omega \cup \mathcal{E}_\omega} u$. From $\ell > r > u$ we obtain $u \xleftrightarrow{\Upsilon\ell}_{\mathcal{R}_\infty \cup \mathcal{E}_\infty} r$ and thus $u \xleftrightarrow{\Upsilon\ell}{}^*_{\mathcal{R}_\omega \cup \mathcal{E}_\omega} r$ follows by (1).

In both cases $\ell \xleftrightarrow{\{\ell\}}{}^*_{\mathcal{R}_\omega \cup \mathcal{E}_\omega} r$ holds. Since $\to_{\mathcal{R}_\omega \cup \mathcal{E}_\omega}$ and $\trianglerighteq$ are closed under contexts and substitutions, the desired inclusion on steps using $\ell \to r$ follows. ◀

We can combine the previous two lemmas to obtain an inclusion in conversions over persistent equations and rules.

▶ **Corollary 24.** *The inclusion* $\xrightarrow[\mathcal{R}_\infty \cup \mathcal{E}_\infty]{S} \subseteq \xleftrightarrow[\mathcal{R}_\omega \cup \mathcal{E}_\omega]{S}{}^*$ *holds for all sets* S *of terms.*

Below, we specialize this result to ground terms.

▶ **Corollary 25.** *If* $s \xleftrightarrow[\mathcal{E}_\infty]{} t$ *for ground terms* s *and* t *then* $s \xleftrightarrow[\mathcal{S}_\omega]{\{s,t\}}{}^* t$.

Proof. We obtain $s \xleftrightarrow{\{s,t\}}{}^*_{\mathcal{R}_\omega \cup \mathcal{E}_\omega} t$ from Corollary 24. Since $>$ is ground-total, all $\mathcal{E}_\omega$ steps in this conversion are $(\mathcal{E}_\omega^{>})^{\pm}$ steps or identities. Hence $s \xleftrightarrow[\mathcal{S}_\omega]{\{s,t\}}{}^* t$ as desired. ◀

The run Γ is called *fair* if the inclusion

$$\mathsf{PCP}_{>}(\mathcal{R}_\omega \cup \mathcal{E}_\omega) \subseteq \bigcup_{i \geqslant 0} \xleftrightarrow[\mathcal{E}_i]{}$$

holds. The following lemma links extended prime critical pairs to standard critical pairs and hence allows us to use results from Section 3 for our main correctness result (Theorem 27 below).

▶ **Lemma 26.** *For a TRS* $\mathcal{R}$ *and a set of equations* $\mathcal{E}$ *the inclusion* $\xleftrightarrow[\mathsf{PCP}(\mathcal{S})]{} \subseteq \xleftrightarrow[\mathsf{PCP}_{>}(\mathcal{R} \cup \mathcal{E})]{} \cup \downarrow_{\mathcal{S}}$ *holds on ground terms.*

Proof. Suppose $s \leftrightarrow_e t$ for ground terms s and t and a prime critical pair $e\colon \ell_2\sigma[r_1\sigma]_p \approx r_2\sigma$ generated from the overlap $\langle \ell_1 \to r_1, p, \ell_2 \to r_2 \rangle$ in $\mathcal{S}$. Let $u_i \approx v_i$ be the equation $\ell_i \approx r_i$ if $\ell_i \to r_i \in \mathcal{R}$ and the equation in $\mathcal{E}^{\pm}$ such that $\ell_i = u_i\tau_i$ and $r_i = v_i\tau_i$ for some substitution τ_i if $\ell_i \to r_i \in \mathcal{E}^{>}$. In the former case we let τ_i be the empty substitution. Since the equations $u_1 \approx v_1$ and $u_2 \approx v_2$ are assumed to be variable-disjoint, the substitution $\tau = \tau_1 \cup \tau_2$ is well-defined. We distinguish two cases.

- If $p \notin \mathcal{P}\mathsf{os}_{\mathcal{F}}(u_2)$ then $\langle u_1 \approx v_1, p, u_2 \approx v_2 \rangle$ is not an overlap and hence $s \downarrow_{\mathcal{S}} t$ by the (extended) Critical Pair Lemma [5].

- Suppose $p \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(u_2)$. Since $u_2|_p\tau\sigma = \ell_2|_p\sigma = \ell_1\sigma = u_1\tau\sigma$ there exist an mgu μ of $u_2|_p$ and u_1, and a substitution ρ such that $\mu\rho = \tau\sigma$. Because $u_i\mu\rho = \ell_i\sigma > r_i\sigma = v_i\mu\rho$, $v_i\mu > u_i\mu$ is impossible. Hence $e'\colon u_2\mu[v_1\mu]_p \approx v_2\mu \in \mathsf{CP}_{>}(\mathcal{R} \cup \mathcal{E})$ and

$$\ell_2\sigma[r_1\sigma]_p = u_2\mu\rho[v_1\mu\rho]_p = u_2\mu[v_1\mu]_p\rho \underset{e'}{\longleftrightarrow} v_2\mu\rho = r_2\sigma$$

  Since e is prime, proper subterms of $\ell_2\sigma|_p = u_2\mu\rho|_p$ are irreducible with respect to $\mathcal{S}$, and hence the same holds for proper subterms of $u_2\mu$. It follows that $e' \in \mathsf{PCP}_{>}(\mathcal{R} \cup \mathcal{E})$ and thus $\ell_2\sigma[r_1\sigma]_p \underset{\mathsf{PCP}_{>}(\mathcal{R}\cup\mathcal{E})}{\longleftrightarrow} r_2\sigma$. Hence also $s \underset{\mathsf{PCP}_{>}(\mathcal{R}\cup\mathcal{E})}{\longleftrightarrow} t$. ◀

This relationship between extended critical pairs among $\mathcal{R} \cup \mathcal{E}$ and critical pairs among $\mathcal{S}$ is the final ingredient for the main result of this section. As in the proof for Knuth-Bendix completion, we establish correctness of ordered completion via source decreasingness.

▶ **Theorem 27.** *If Γ is fair then $\mathcal{S}_\omega$ is ground-complete and $\overset{*}{\underset{\mathcal{E}_0}{\longleftrightarrow}} = \overset{*}{\underset{\mathcal{R}_\omega\cup\mathcal{E}_\omega}{\longleftrightarrow}}$.*

Proof. Termination of $\mathcal{S}_\omega$ is a consequence of Lemma 21 and the definition of $\mathcal{E}_\omega^{>}$. Next we show that $\mathcal{S}_\omega$ is ground-confluent. To this end, we show that labeled $\mathcal{S}_\omega$ reduction is source decreasing on ground terms. So let s, t, and u be ground terms such that $t \,{}_{\mathcal{S}_\omega}\!\overset{s}{\leftarrow} s \overset{s}{\rightarrow}_{\mathcal{S}_\omega} u$. From Lemma 6 we obtain $t \mathrel{\triangledown_s^2} u$ (where $\mathcal{S}_\omega$ takes the place of $\mathcal{R}$ in the definition of $\triangledown_s$). Let $v \mathrel{\triangledown_s} w$ appear in this sequence (so $t = v$ or $w = u$ and both terms are ground). We have $s > v$, $s > w$, and

$$(v, w) \in {\downarrow_{\mathcal{S}_\omega}} \cup \bigcup_{i\geqslant 0} \underset{\mathcal{E}_i}{\longleftrightarrow}$$

by the definition of $\triangledown_s$, Lemma 26, and fairness of Γ.

- If $v \downarrow_{\mathcal{S}_\omega} w$ then $v \overset{\mathbb{V}v}{\underset{\mathcal{S}_\omega}{\longrightarrow}}{}^{*} \cdot {}^{*}_{\mathcal{S}_\omega}\overset{\mathbb{V}w}{\longleftarrow} w$ and thus $v \overset{\vee s}{\underset{\mathcal{S}_\omega}{\longleftrightarrow}}{}^{*} w$.
- If $v \leftrightarrow_{\mathcal{E}_i} w$ for some $i \geqslant 0$ then $v \overset{\vee s}{\underset{\mathcal{S}_\omega}{\longleftrightarrow}}{}^{*} w$ by Corollary 25.

Hence $t \overset{\vee s}{\underset{\mathcal{S}_\omega}{\longleftrightarrow}}{}^{*} u$. Confluence of the ARS that is obtained by restricting $\mathcal{S}_\omega$ to ground terms now follows from Lemma 4. It remains to show $\overset{*}{\underset{\mathcal{E}_0}{\longleftrightarrow}} = \overset{*}{\underset{\mathcal{R}_\omega\cup\mathcal{E}_\omega}{\longleftrightarrow}}$. Using Corollary 20 we obtain $\rightarrow_{\mathcal{E}_i\cup\mathcal{R}_i} \subseteq \overset{*}{\underset{\mathcal{E}_0}{\longleftrightarrow}}$ for all i by a straightforward induction argument. This in turn yields $\overset{*}{\underset{\mathcal{E}_\infty\cup\mathcal{R}_\infty}{\longleftrightarrow}} \subseteq \overset{*}{\underset{\mathcal{E}_0}{\longleftrightarrow}}$ and in particular $\overset{*}{\underset{\mathcal{E}_\omega\cup\mathcal{R}_\omega}{\longleftrightarrow}} \subseteq \overset{*}{\underset{\mathcal{E}_0}{\longleftrightarrow}}$. The reverse inclusion follows from Corollary 24 and the inclusion $\overset{*}{\underset{\mathcal{E}_0}{\longleftrightarrow}} \subseteq \overset{*}{\underset{\mathcal{E}_\infty\cup\mathcal{R}_\infty}{\longleftrightarrow}}$. ◀

5 Canonicity

In this section we revisit Métivier work [13], aiming at generalizing his uniqueness result for canonical TRSs and at establishing a transformation to simplify ground-complete TRSs. A key notion is normalization equivalence.

▶ **Definition 28.** Two ARSs $\mathcal{A}$ and $\mathcal{B}$ are said to be *(conversion) equivalent* if $\overset{*}{\underset{\mathcal{A}}{\longleftrightarrow}} = \overset{*}{\underset{\mathcal{B}}{\longleftrightarrow}}$ and *normalization equivalent* if $\overset{!}{\underset{\mathcal{A}}{\rightarrow}} = \overset{!}{\underset{\mathcal{B}}{\rightarrow}}$.

The following example shows that the two equivalence notions defined above are different.

▶ **Example 29.** Consider the following ARSs:

While $\mathcal{A}_1$ and $\mathcal{B}_1$ are conversion equivalent but not normalization equivalent, the ARSs $\mathcal{A}_2$ and $\mathcal{B}_2$ are normalization equivalent but not conversion equivalent.

The easy proof (by induction on the length of conversions) of the following result is omitted.

▶ **Lemma 30.** *Normalization equivalent terminating ARSs are equivalent.* ◀

Note that the termination assumption can be weakened to weak normalization. However, the present version suffices to prove the following lemma that we employ in our proof of Métivier's transformation result [13] (Theorem 37 below).

▶ **Lemma 31.** *Let $\mathcal{A}$ and $\mathcal{B}$ be ARSs such that $\mathsf{NF}(\mathcal{B}) \subseteq \mathsf{NF}(\mathcal{A})$ and either*
1. *$\to_{\mathcal{B}} \subseteq \to_{\mathcal{A}}^+$ or*
2. *$\to_{\mathcal{B}} \subseteq \leftrightarrow_{\mathcal{A}}^*$ and $\mathcal{B}$ is terminating.*

If $\mathcal{A}$ is complete then $\mathcal{B}$ is complete and normalization equivalent to $\mathcal{A}$.

Proof. We first show $\to_{\mathcal{B}}^! \subseteq \to_{\mathcal{A}}^!$. In case (a), from the inclusion $\to_{\mathcal{B}} \subseteq \to_{\mathcal{A}}^+$ we infer that $\mathcal{B}$ is terminating. Moreover, $\to_{\mathcal{B}}^* \subseteq \to_{\mathcal{A}}^*$ and, since $\mathsf{NF}(\mathcal{B}) \subseteq \mathsf{NF}(\mathcal{A})$, also $\to_{\mathcal{B}}^! \subseteq \to_{\mathcal{A}}^!$. For case (b), $\to_{\mathcal{B}}^! \subseteq \to_{\mathcal{A}}^!$ holds because $\to_{\mathcal{B}}^! \subseteq \leftrightarrow_{\mathcal{A}}^*$, so by confluence of $\mathcal{A}$ and $\mathsf{NF}(\mathcal{B}) \subseteq \mathsf{NF}(\mathcal{A})$ we obtain $\to_{\mathcal{B}}^! \subseteq \to_{\mathcal{A}}^!$. Next we show that the reverse inclusion $\to_{\mathcal{A}}^! \subseteq \to_{\mathcal{B}}^!$ holds in both cases. Let $a \to_{\mathcal{A}}^! b$. Because $\mathcal{B}$ is terminating, $a \to_{\mathcal{B}}^! c$ for some $c \in \mathsf{NF}(\mathcal{B})$. So $a \to_{\mathcal{A}}^! c$ and thus $b = c$ from the confluence of $\mathcal{A}$. It follows that $\mathcal{A}$ and $\mathcal{B}$ are normalization equivalent. It remains to show that $\mathcal{B}$ is locally confluent. This follows from the sequence of inclusions

$$ {}_{\mathcal{B}}\!\leftarrow \cdot \to_{\mathcal{B}} \;\subseteq\; \leftrightarrow_{\mathcal{A}}^* \;\subseteq\; \to_{\mathcal{A}}^! \cdot {}_{\mathcal{A}}^{!}\!\leftarrow \;\subseteq\; \to_{\mathcal{B}}^! \cdot {}_{\mathcal{B}}^{!}\!\leftarrow $$

where we obtain the inclusions from $\to_{\mathcal{B}} \subseteq \leftrightarrow_{\mathcal{A}}^*$, confluence of $\mathcal{A}$, termination of $\mathcal{A}$, and normalization equivalence of $\mathcal{A}$ and $\mathcal{B}$, respectively. ◀

In the above lemma, completeness can be weakened to semi-completeness (i.e., the combination of confluence and weak normalization), which is not true for Theorem 37 as shown by Gramlich [7]. Again, the present version suffices for our purposes.

▶ **Definition 32.** A TRS $\mathcal{R}$ is *left-reduced* if $\ell \in \mathsf{NF}(\mathcal{R} \setminus \{\ell \to r\})$ for every rewrite rule $\ell \to r$ in $\mathcal{R}$. We say that $\mathcal{R}$ is *right-reduced* if $r \in \mathsf{NF}(\mathcal{R})$ for every rewrite rule $\ell \to r$ in $\mathcal{R}$. A *reduced* TRS is left- and right-reduced. A reduced complete TRS is called *canonical*.

Theorem 37 below states that we can always eliminate redundancy in a complete TRS. This is achieved by the two-stage transformation defined below.

▶ **Definition 33.** Two TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$ over the same signature $\mathcal{F}$ are called *literally similar*, denoted by $\mathcal{R}_1 \doteq \mathcal{R}_2$, if every rewrite rule in $\mathcal{R}_1$ has a variant in $\mathcal{R}_2$ and vice-versa.

Given a TRS $\mathcal{R}$, we write $\mathcal{R}/\doteq$ for a set of representatives of the equivalence classes of rules in $\mathcal{R}$ with respect to $\doteq$. In other words, $\mathcal{R}/\doteq$ is a variant-free version of $\mathcal{R}$.

▶ **Definition 34.** Given a terminating TRS $\mathcal{R}$, the TRSs $\dot{\mathcal{R}}$ and $\ddot{\mathcal{R}}$ are defined as follows:

$$\dot{\mathcal{R}} = \{\ell \to r{\downarrow_{\mathcal{R}}} \mid \ell \to r \in \mathcal{R}\}/\doteq,$$
$$\ddot{\mathcal{R}} = \{\ell \to r \in \dot{\mathcal{R}} \mid \ell \in \mathsf{NF}(\dot{\mathcal{R}} \setminus \{\ell \to r\})\}.$$

Here $t{\downarrow_{\mathcal{R}}}$ stands for an arbitrary but fixed normal form of t.

The TRS $\dot{\mathcal{R}}$ is obtained from $\mathcal{R}$ by first normalizing the right-hand sides and then taking representatives of variants of the resulting rules, thereby making sure that the result does not contain several variants of the same rule. To obtain $\ddot{\mathcal{R}}$ we remove the rules of $\dot{\mathcal{R}}$ whose left-hand sides are reducible with another rule of $\dot{\mathcal{R}}$.

The following example shows why the result of $\dot{\mathcal{R}}$ has to be variant-free.

▶ **Example 35.** Consider the TRS $\mathcal{R}_1$ consisting of the four rules

$$\mathsf{f}(x) \to \mathsf{a}\,, \qquad \mathsf{f}(y) \to \mathsf{b}\,, \qquad \mathsf{a} \to \mathsf{c}\,, \qquad \mathsf{b} \to \mathsf{c}\,.$$

Then the first transformation without taking representatives of rules would yield $\dot{\mathcal{R}}_1$

$$\mathsf{f}(x) \to \mathsf{c}\,, \qquad \mathsf{f}(y) \to \mathsf{c}\,, \qquad \mathsf{a} \to \mathsf{c}\,, \qquad \mathsf{b} \to \mathsf{c}\,.$$

and the second one $\ddot{\mathcal{R}}_1$

$$\mathsf{a} \to \mathsf{c}\,, \qquad \mathsf{b} \to \mathsf{c}\,.$$

Note that $\ddot{\mathcal{R}}_1$ is *not* equivalent to $\mathcal{R}_1$. This is caused by the fact that the result of the first transformation was no longer variant-free.

The following result is folklore; the proof has been formalized [8].

▶ **Lemma 36.** *Two terms s and t are variants if and only if $s \doteq t$.*

▶ **Theorem 37.** *If $\mathcal{R}$ is a complete TRS then $\ddot{\mathcal{R}}$ is a normalization and conversion equivalent canonical TRS.*

The proof by Métivier [13, Theorem 7] is hard to reconstruct. The proof in [16, Exercise 7.4.7] involves 13 steps with lots of redundancy. Our proof below proceeds by induction on the well-founded encompassment order $\trianglerighteq$.

Proof. Let $\mathcal{R}$ be a complete TRS. The inclusions $\ddot{\mathcal{R}} \subseteq \dot{\mathcal{R}} \subseteq \to_{\mathcal{R}}^{+}$ are obvious from the definitions. Since $\mathcal{R}$ and $\dot{\mathcal{R}}$ have the same left-hand sides, their normal forms coincide. We show that $\mathsf{NF}(\ddot{\mathcal{R}}) \subseteq \mathsf{NF}(\dot{\mathcal{R}})$. To this end we show that $\ell \notin \mathsf{NF}(\ddot{\mathcal{R}})$ whenever $\ell \to r \in \dot{\mathcal{R}}$ by induction on ℓ with respect to the well-founded order $\trianglerighteq$. If $\ell \to r \in \ddot{\mathcal{R}}$ then $\ell \notin \mathsf{NF}(\ddot{\mathcal{R}})$ holds. So suppose $\ell \to r \notin \ddot{\mathcal{R}}$. By definition of $\ddot{\mathcal{R}}$, $\ell \notin \mathsf{NF}(\dot{\mathcal{R}} \setminus \{\ell \to r\})$. So there exists a rewrite rule $\ell' \to r' \in \dot{\mathcal{R}}$ different from $\ell \to r$ such that $\ell \trianglerighteq \ell'$. We distinguish two cases.

- If $\ell \triangleright \ell'$ then we obtain $\ell' \notin \mathsf{NF}(\ddot{\mathcal{R}})$ from the induction hypothesis and hence $\ell \notin \mathsf{NF}(\ddot{\mathcal{R}})$ as desired.
- If $\ell \doteq \ell'$ then by Lemma 36 there exists a renaming σ such that $\ell = \ell'\sigma$. Since $\dot{\mathcal{R}}$ is right-reduced by construction, r and r' are normal forms of $\dot{\mathcal{R}}$. The same holds for $r'\sigma$ because normal forms are closed under renaming. We have $r \,{}_{\dot{\mathcal{R}}}\!\leftarrow \ell = \ell'\sigma \to_{\dot{\mathcal{R}}} r'\sigma$. Since $\dot{\mathcal{R}}$ is confluent as a consequence of Lemma 31(1), $r = r'\sigma$. Hence $\ell' \to r'$ is a variant of $\ell \to r$, contradicting the assumption that TRSs are variant-free.

From Lemma 31(1) we infer that the TRSs $\dot{\mathcal{R}}$ and $\ddot{\mathcal{R}}$ are complete and normalization equivalent to $\mathcal{R}$. The TRS $\ddot{\mathcal{R}}$ is right-reduced because $\ddot{\mathcal{R}} \subseteq \dot{\mathcal{R}}$ and $\dot{\mathcal{R}}$ is right-reduced. From $\mathsf{NF}(\ddot{\mathcal{R}}) = \mathsf{NF}(\dot{\mathcal{R}})$ we easily infer that $\ddot{\mathcal{R}}$ is left-reduced. It follows that $\ddot{\mathcal{R}}$ is canonical. It remains to show that $\ddot{\mathcal{R}}$ is not only normalization equivalent but also (conversion) equivalent to $\mathcal{R}$. This is an immediate consequence of Lemma 30. ◀

For our next result we need the following technical lemma.

▶ **Lemma 38.** *Let $\mathcal{R}$ be a right-reduced TRS and let s be a reducible term which is minimal with respect to $\rhd$. If $s \to^+_{\mathcal{R}} t$ then $s \to t$ is a variant of a rule in $\mathcal{R}$.*

Proof. Let $\ell \to r$ be the rewrite rule that is used in the first step from s to t. So $s \trianglerighteq \ell$. By assumption, $s \rhd \ell$ does not hold and thus $s \doteq \ell$. According to Lemma 36 there exists a renaming σ such that $s = \ell\sigma$. We have $s \to_{\mathcal{R}} r\sigma \to^*_{\mathcal{R}} t$. Because $\mathcal{R}$ is right-reduced, $r \in \mathsf{NF}(\mathcal{R})$. Since normal forms are closed under renaming, also $r\sigma \in \mathsf{NF}(\mathcal{R})$ and thus $r\sigma = t$. It follows that $s \to t$ is a variant of $\ell \to r$. ◀

In our formalization, the above proof is the first spot where we actually need that $\mathcal{R}$ satisfies the variable condition (more precisely, right-hand sides of rules do not introduce fresh variables). The next result is the main result of this section.

▶ **Theorem 39.** *Normalization equivalent reduced TRSs are unique up to literal similarity.*

Proof. Let $\mathcal{R}$ and $\mathcal{S}$ be normalization equivalent reduced TRSs. Suppose $\ell \to r \in \mathcal{R}$. Because $\mathcal{R}$ is right-reduced, $r \in \mathsf{NF}(\mathcal{R})$ and thus $\ell \neq r$. Hence $\ell \to^+_{\mathcal{S}} r$ by normalization equivalence. Because $\mathcal{R}$ is left-reduced, ℓ is a minimal (with respect to $\rhd$) $\mathcal{R}$-reducible term. Another application of normalization equivalence yields that ℓ is minimal $\mathcal{S}$-reducible. Hence $\ell \to r$ is a variant of a rule in $\mathcal{S}$ by Lemma 38. ◀

▶ **Example 40.** Consider the rewrite system $\mathcal{R}$ of combinatory logic with equality test, studied by Klop [11]:

$$\mathsf{S}xyz \to xz(yz)\,, \qquad \mathsf{K}xy \to x\,, \qquad \mathsf{I}x \to x\,, \qquad \mathsf{D}xx \to \mathsf{E}\,.$$

The rewrite system $\mathcal{R}$ is reduced, but neither terminating nor confluent. One might ask whether there is another reduced rewrite system that computes the same normal forms for every starting term. Theorem 39 shows that $\mathcal{R}$ is unique up to variable renaming.

We show that the corresponding result of Métivier [13, Theorem 8] is an easy consequence of Theorem 39. Here a TRS $\mathcal{R}$ is said to be compatible with a reduction order $>$ if $\ell > r$ for every rewrite rule $\ell \to r$ of $\mathcal{R}$.

▶ **Theorem 41.** *Let $\mathcal{R}$ and $\mathcal{S}$ be equivalent canonical TRSs. If $\mathcal{R}$ and $\mathcal{S}$ are compatible with the same reduction order then $\mathcal{R} \doteq \mathcal{S}$.*

Proof. Suppose $\mathcal{R}$ and $\mathcal{S}$ are compatible with the reduction order $>$. We show that $\to^!_{\mathcal{R}} \subseteq \to^!_{\mathcal{S}}$. Let $s \to^!_{\mathcal{R}} t$. We show that $t \in \mathsf{NF}(\mathcal{S})$. Let u be the unique $\mathcal{S}$-normal form of t. We have $t \to^!_{\mathcal{S}} u$ and thus $t \leftrightarrow^*_{\mathcal{R}} u$ because $\mathcal{R}$ and $\mathcal{S}$ are equivalent. Since $t \in \mathsf{NF}(\mathcal{R})$, we have $u \to^!_{\mathcal{R}} t$. If $t \neq u$ then both $t > u$ (as $t \to^!_{\mathcal{S}} u$) and $u > t$ (as $u \to^!_{\mathcal{R}} t$), which is impossible. Hence $t = u$ and thus $t \in \mathsf{NF}(\mathcal{S})$. Together with $s \leftrightarrow^*_{\mathcal{S}} t$, which follows from the equivalence of $\mathcal{R}$ and $\mathcal{S}$, we conclude that $s \to^!_{\mathcal{S}} t$. We obtain $\to^!_{\mathcal{S}} \subseteq \to^!_{\mathcal{R}}$ by symmetry. Hence $\mathcal{R}$ and $\mathcal{S}$ are normalization equivalent and the result follows from Theorem 39. ◀

The final result in this section is in the spirit of Theorem 37 but for ordered completion, showing that a ground-complete system can be interreduced to some extent. Let $>$ again be a ground-total reduction order.

▶ **Definition 42.** Given a ground-complete system $\mathcal{S} = \mathcal{R} \cup \mathcal{E}^>$, we define

$$\mathcal{R}' = \{\ell \to r \mid \ell \to r \in \dot{\mathcal{Q}} \text{ and } \ell \in \mathsf{NF}(\xrightarrow{\rhd}_{\mathcal{S}})\}\,,$$
$$\mathcal{E}' = \{s{\downarrow_{\mathcal{R}'}} \approx t{\downarrow_{\mathcal{R}'}} \mid s \approx t \in \mathcal{E}\} \setminus =\,,$$

where $\mathcal{Q} = \mathcal{R} \cup (\mathcal{E}^{\pm} \cap >)$.

Here we write $t \xrightarrow{\rhd}_{\mathcal{S}} u$ if there are a rule $\ell \to r \in \mathcal{S}$, a context C, and a substitution σ such that $t = C[\ell\sigma]$, $u = C[r\sigma]$, and $t \rhd \ell$. For example, if $\mathcal{E}$ is empty and $\mathcal{R}$ consists of the single rule $\mathsf{f}(x,y) \to \mathsf{g}(x)$ we have $\mathsf{f}(y,z) \in \mathsf{NF}(\xrightarrow{\rhd}_{\mathcal{S}})$, but $\mathsf{f}(\mathsf{g}(x),y) \notin \mathsf{NF}(\xrightarrow{\rhd}_{\mathcal{S}})$ and $\mathsf{f}(x,x) \notin \mathsf{NF}(\xrightarrow{\rhd}_{\mathcal{S}})$.

▶ **Theorem 43.** *If $\mathcal{S} = \mathcal{R} \cup \mathcal{E}^{>}$ is ground-complete then $\mathcal{S}' = \mathcal{R}' \cup \mathcal{E}'^{>}$ is ground-complete and normalization and conversion equivalent on ground terms.*

Proof. We first show $\mathsf{NF}(\mathcal{S}') \subseteq \mathsf{NF}(\mathcal{S})$. For a rule $\ell \to r \in \mathcal{S}$, let $b_{\ell\to r}$ be $\bot$ if $\ell \to r \in \mathcal{Q}$ and $\top$ otherwise. We prove $\ell \notin \mathsf{NF}(\mathcal{S}')$ for every rule $\ell \to r \in \mathcal{S}$, by induction on $(\ell, b_{\ell\to r})$ with respect to the lexicographic combination of $\rhd$ and the order where $\top > \bot$.

- If $\ell \to r \in \mathcal{Q}$ two cases can be distinguished. If $\ell \notin \mathsf{NF}(\xrightarrow{\rhd}_{\mathcal{S}})$ then $\ell \rhd \ell'$ for some rule $\ell' \to r' \in \mathcal{S}$ and thus $\ell' \notin \mathsf{NF}(\mathcal{S}')$ by the induction hypothesis. Hence also $\ell \notin \mathsf{NF}(\mathcal{S}')$. If $\ell \in \mathsf{NF}(\xrightarrow{\rhd}_{\mathcal{S}})$ then, by construction of $\mathcal{R}'$, there is some rule $\ell \to r' \in \mathcal{R}'$ (modulo renaming), so $\ell \notin \mathsf{NF}(\mathcal{S}')$.
- If $\ell \to r \notin \mathcal{Q}$ then $\ell = u\sigma$ and $r = v\sigma$ for some equation $u \approx v \in \mathcal{E}^{\pm}$ and substitution σ such that $\ell > r$. We distinguish two cases. First, if $u \in \mathsf{NF}(\mathcal{R}')$ then $u = u{\downarrow_{\mathcal{R}'}}$. We have $\ell > r \geqslant v{\downarrow_{\mathcal{R}'}}\sigma$ because $\mathcal{R}' \subseteq >$ and hence $u \neq v{\downarrow_{\mathcal{R}'}}$. It follows that $u \approx v{\downarrow_{\mathcal{R}'}} \in \mathcal{E}'^{\pm}$ and thus $\ell \to v{\downarrow_{\mathcal{R}'}}\sigma \in \mathcal{E}'^{\succ}$. Hence $\ell \notin \mathsf{NF}(\mathcal{S}')$. Second, if $u \notin \mathsf{NF}(\mathcal{R}')$ then $u \notin \mathsf{NF}(\dot{\mathcal{Q}})$ since $\mathcal{R}' \subseteq \dot{\mathcal{Q}}$. So there exists a rule $\ell' \to r' \in \mathcal{Q}$ such that $u \trianglerighteq \ell'$. Clearly $\ell \trianglerighteq \ell'$. Since $\ell \to r \notin \mathcal{Q}$, the induction hypothesis yields $\ell' \notin \mathsf{NF}(\mathcal{S}')$. Hence also $\ell \notin \mathsf{NF}(\mathcal{S}')$.

We next establish the inclusion $\to_{\mathcal{S}'} \subseteq \leftrightarrow^*_{\mathcal{S}}$ on ground terms. We have $\mathcal{R}' \cup \mathcal{E}' \subseteq \leftrightarrow^*_{\mathcal{R}\cup\mathcal{E}}$ by construction. For ground terms s and t, a step $s \to_{\mathcal{S}'} t$ implies $s \leftrightarrow_{\mathcal{R}'\cup\mathcal{E}'} t$ and hence existence of a conversion $s \leftrightarrow^*_{\mathcal{R}\cup\mathcal{E}} t$. We can also obtain such a conversion where all intermediate terms are ground by replacing every variable with some ground term. Since the reduction order $>$ is ground-total, $\to_{\mathcal{R}\cup\mathcal{E}} \subseteq \leftrightarrow^{=}_{\mathcal{S}}$ holds on ground terms. Hence there is a conversion $s \leftrightarrow^*_{\mathcal{S}} t$.

Moreover, the system $\mathcal{S}'$ is clearly terminating as it is included in $>$. Thus the result follows from Lemma 31(2), viewing $\mathcal{S}$ and $\mathcal{S}'$ as ARSs on ground terms. ◀

We illustrate the transformation of Definition 42 on a concrete example.

▶ **Example 44.** Consider the following system with $\mathcal{R}$ consisting of one rule and $\mathcal{E}$ consisting of three equations:

$$\mathsf{s}(\mathsf{s}(x)) + \mathsf{s}(x) \to \mathsf{s}(x) + \mathsf{s}(\mathsf{s}(x))\,, \qquad x + \mathsf{s}(y) \approx \mathsf{s}(x+y)\,, \qquad x + y \approx y + x\,,$$
$$\mathsf{s}(x) + y \approx \mathsf{s}(x+y)\,.$$

It is ground-complete for the lexicographic path order [9] with $+ > \mathsf{s}$ as precedence. We have $\mathcal{Q} = \mathcal{R} \cup \{x + \mathsf{s}(y) \to \mathsf{s}(x+y),\ \mathsf{s}(x) + y \to \mathsf{s}(x+y)\}$. Since the term $\mathsf{s}(\mathsf{s}(x)) + \mathsf{s}(x)$ is reducible by the rule $\mathsf{s}(x) + x \to x + \mathsf{s}(x) \in \mathcal{S}$ and $\mathsf{s}(\mathsf{s}(x)) + \mathsf{s}(x) \rhd \mathsf{s}(x) + x$, the rule of $\mathcal{R}$ does not remain in $\mathcal{R}'$. Hence, $\mathcal{R}' = \{x + \mathsf{s}(y) \to \mathsf{s}(x+y),\ \mathsf{s}(x) + y \to \mathsf{s}(x+y)\}$ and $\mathcal{E}' = \{x + y \approx y + x\}$.

One may wonder whether $\mathcal{R}'$ can simply be defined as $\ddot{\mathcal{Q}}$ instead of imposing a strict encompassment condition. The following example shows that this destroys reducibility.

▶ **Example 45.** Consider the following system where $\mathcal{R}$ consists of two rules and $\mathcal{E}$ consists of one equation:

$$\mathsf{f}(x,y) \to \mathsf{g}(x)\,, \qquad \mathsf{f}(x,y) \to \mathsf{g}(y)\,, \qquad \mathsf{g}(x) \approx \mathsf{g}(y)\,.$$

Then $\mathcal{R} \cup \mathcal{E}^{>}$ is ground-complete if $>$ is the lexicographic path order with $\mathsf{f} > \mathsf{g}$ as precedence. We have $\mathcal{R}' = \dot{\mathcal{Q}} = \mathcal{Q} = \mathcal{R}$ and $\mathcal{E}' = \mathcal{E}$ but $\ddot{\mathcal{Q}} = \varnothing$.

Note that we obtain an equivalent ground-complete system if we add, for instance, an equation $\mathsf{g}(\mathsf{g}(x)) \approx \mathsf{g}(y)$. This shows that even systems which are simplified with respect to the procedure suggested by Theorem 43 are not unique.

6 Conclusion

We gave new and concise correctness proofs of Knuth-Bendix and ordered completion. These results specifically apply to infinite runs, a case in which the reasoning becomes more tedious as the encompassment condition for the collapse rule is essential. We also contributed new results about canonicity, related to normalization and conversion equivalence. In particular we generalized the distinguished theorem by Métivier on uniqueness of canonical rewrite systems. All our results are formalized in IsaFoR.

As future work, we want to extend our proofs and formalization to cover completeness results for both Knuth-Bendix and ordered completion [2, 5]. Furthermore, we will apply our techniques to AC completion and the decidable case of ground completion.

Acknowledgements. We thank the anonymous reviewers for their comments to improve the presentation of the paper.

References

1 F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

2 L. Bachmair. *Canonical Equational Proofs*. Birkhäuser, 1991.

3 L. Bachmair and N. Dershowitz. Commutation, transformation, and termination. In *Proc. 8th International Conference on Automated Deduction*, volume 230 of *Lecture Notes in Computer Science*, pages 5–20, 1986. doi:10.1007/3-540-16780-3_76.

4 L. Bachmair, N. Dershowitz, and J. Hsiang. Orderings for equational proofs. In *Proc. 1st IEEE Symposium on Logic in Computer Science*, pages 346–357, 1986.

5 L. Bachmair, N. Dershowitz, and D. A. Plaisted. Completion without failure. In H. Aït Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 2: Rewriting Techniques of *Progress in Theoretical Computer Science*, pages 1–30. Academic Press, 1989.

6 S. Burckel. Syntactical methods for braids of three strands. *Journal of Symbolic Computation*, 31:557–564, 2001. doi:10.1006/jsco.2000.0473.

7 B. Gramlich. On interreduction of semi-complete term rewriting systems. *Theoretical Computer Science*, 258(1-2):435–451, 2001. doi:10.1016/S0304-3975(00)00030-X.

8 N. Hirokawa, A. Middeldorp, and C. Sternagel. A new and formalized proof of abstract completion. In *Proc. 5th International Conference on Interactive Theorem Proving*, volume 8558 of *Lecture Notes in Computer Science*, pages 292–307, 2014. doi:10.1007/978-3-319-08970-6_19.

9 S. Kamin and J. J. Lévy. Two generalizations of the recursive path ordering. Unpublished manuscript, University of Illinois, 1980.

10 D. Kapur and P. Narendran. A finite Thue system with decidable word problem and without equivalent finite canonical system. *Theoretical Computer Science*, 35:337–344, 1985. doi:10.1016/0304-3975(85)90023-4.

11 J. W. Klop. *Combinatory Reduction Systems*. PhD thesis, Utrecht University, 1980.

12 D. E. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. 1970.

13 Y. Métivier. About the rewriting systems produced by the Knuth-Bendix completion algorithm. *Information Processing Letters*, 16(1):31–34, 1983. doi:10.1016/0020-0190(83)90009-1.

14 T. Nipkow, L. C. Paulson, and M. 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 C. Sternagel and R. Thiemann. Formalizing Knuth-Bendix orders and Knuth-Bendix completion. In *Proc. 24th International Conference on Rewriting Techniques and Applications*, volume 21 of *Leibniz International Proceedings in Informatics*, pages 287–302. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi: 10.4230/LIPIcs.RTA.2013.287. doi:10.4230/LIPIcs.RTA.2013.287.

16 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

17 V. van Oostrom. Confluence by decreasing diagrams – converted. In *Proc. 19th International Conference on Rewriting Techniques and Applications*, volume 5117 of *Lecture Notes in Computer Science*, pages 306–320, 2008. doi:10.1007/978-3-540-70590-1_21.

18 F. Winkler and B. Buchberger. A criterion for eliminating unnecessary reductions in the Knuth-Bendix algorithm. In *Proc. Colloquium on Algebra, Combinatorics and Logic in Computer Science, Vol. II*, volume 42 of *Colloquia Mathematica Societatis J. Bolyai*, pages 849–869, 1986.

Refutation of Sallé's Longstanding Conjecture

Benedetto Intrigila[1], Giulio Manzonetto[2], and Andrew Polonsky[3]

1 **Dipartimento di Ingegneria dell'Impresa, Università di Roma Tor Vergata, Rome, Italy**
2 **CNRS, IRIF, UMR 8243, Université Paris-Diderot, Sorbonne Paris Cité, Paris, France; and LIPN, UMR 7030, Université Paris 13, Sorbonne Paris Cité, Villetaneuse, France**
3 **CNRS, IRIF, UMR 8243, Université Paris-Diderot, Sorbonne Paris Cité, Paris, France**

Abstract

The λ-calculus possesses a strong notion of extensionality, called "the ω-rule", which has been the subject of many investigations. It is a longstanding open problem whether the equivalence obtained by closing the theory of Böhm trees under the ω-rule is strictly included in Morris's original observational theory, as conjectured by Sallé in the seventies. In a recent work, Breuvart *et al.* have shown that Morris's theory satisfies the ω-rule. In this paper we demonstrate that the two aforementioned theories actually coincide, thus disproving Sallé's conjecture.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Lambda calculus, observational equivalence, Böhm trees, ω-rule

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.20

1 Introduction

The problem of determining when two programs are equivalent is central in computer science. For instance, it is necessary to verify that the optimizations performed by a compiler actually preserve the meaning of the program. For λ-calculi, it has become standard to consider two λ-terms M and N as equivalent when they are *contextually equivalent* with respect to some fixed set $\mathcal{O}$ of observables [22]. This means that it is possible to plug either M or N into any context $C[-]$ without noticing any difference in the global behaviour: $C[M]$ produces a result belonging to $\mathcal{O}$ exactly when $C[N]$ does. The problem of working with this definition, is that the quantification over all possible contexts is difficult to handle. Therefore, many researchers undertook a quest for characterizing observational equivalences both semantically, by defining fully abstract denotational models, and syntactically, by comparing possibly infinite trees representing the programs executions.

The most famous observational equivalence is obtained by considering as observables the head normal forms, which are λ-terms representing stable amounts of information coming out of the computation. Introduced by Hyland [13] and Wadsworth [30], it has been ubiquitously studied in the literature [2, 11, 9, 25, 19, 4], since it enjoys many interesting properties. By definition, it corresponds to the extensional λ-theory $\mathcal{H}^*$ which is the greatest consistent sensible λ-theory [2, Thm. 16.2.6]. Semantically, it arises as the λ-theory of Scott's pioneering model $\mathcal{D}_\infty$ [27], a result which first appeared in [13] and [30]. Recently, Breuvart provided a characterization of all K-models that are fully abstract for $\mathcal{H}^*$ [4]. As shown in [2, Thm. 16.2.7], two λ-terms are equivalent in $\mathcal{H}^*$ exactly when their Böhm trees are equal up to countably many (possibly) infinite η-expansions.

However, the head normal forms are not the only reasonable choice of observables. For instance, the original extensional contextual equivalence defined by Morris in [22] arises by considering as observables the β-normal forms, that represent completely defined results. We denote by $\mathcal{H}^+$ the λ-theory corresponding to Morris's observational equivalence (using the notation of [20, 5], while it is denoted by $\mathscr{T}_{\mathrm{NF}}$ in [2]). The λ-theory $\mathcal{H}^+$ is sensible and distinct from $\mathcal{H}^*$, so $\mathcal{H}^+ \subsetneq \mathcal{H}^*$. Despite the fact that the equality in $\mathcal{H}^+$ has been the subject of fewer investigations, it has been characterized both semantically and syntactically. In [8], Coppo *et al.* proved that $\mathcal{H}^+$ corresponds to the λ-theory induced by a suitable filter model. More recently, Manzonetto and Ruoppolo introduced a simpler model of $\mathcal{H}^+$ living in the relational semantics [20] and Breuvart *et al.* provided necessary and sufficient conditions for a relational model to be fully abstract for $\mathcal{H}^+$ [5]. From a syntactic perspective, Hyland proved in [12] that two λ-terms are equivalent in $\mathcal{H}^+$ exactly when their Böhm trees are equal up to countably many η-expansions of finite size (cf. [25, §11.2] and [17]).

We have seen that both observational equivalences correspond to some extensional equalities between Böhm trees. A natural question is whether $\mathcal{H}^+$ can be generated just by adding the η-conversion to the λ-theory $\mathcal{B}$ equating all λ-terms having the same Böhm tree. The λ-theory $\mathcal{B}\eta$ so defined has been little studied in the literature, probably because it does not arise as an observational equivalence nor is induced by some known denotational model. In [2, Lemma 16.4.3], Barendregt shows that one η-expansion in a λ-term M can generate infinitely many finite η-expansions on its Böhm tree $\mathrm{BT}(M)$. In [2, Lemma 16.4.4], he exhibits two λ-terms that are equal in $\mathcal{H}^+$ but distinct in $\mathcal{B}\eta$, thus proving that $\mathcal{B}\eta \subsetneq \mathcal{H}^+$.

However, the λ-calculus also possesses another notion of extensionality, known as the *ω-rule*, which is strictly stronger than η-conversion. Such a rule has been studied by many researchers in connection with several λ-theories [16, 1, 23, 3, 15]. Formally, the ω-rule states that for all λ-terms M and N, $M = N$ whenever $MP = NP$ holds for all closed λ-terms P. A λ-theory $\mathcal{T}$ *satisfies the ω-rule* whenever it is closed under such a rule. Since this is such an impredicative rule, we can meaningfully wonder how the λ-theory $\mathcal{B}\omega$, obtained as the closure of $\mathcal{B}$ under the ω-rule, compares with the other λ-theories. As shown by Barendregt in [2, Lemma 16.4.4], $\mathcal{B}\eta$ does not satisfy the ω-rule, while $\mathcal{H}^*$ does [2, Thm. 17.2.17].

Therefore, the two possible scenarios are the following:

$$\mathcal{B}\eta \subsetneq \mathcal{B}\omega \subseteq \mathcal{H}^+ \subsetneq \mathcal{H}^* \qquad \text{or} \qquad \mathcal{B}\eta \begin{array}{c} \subsetneq \\ \not\supseteq \\ \not\subseteq \end{array} \begin{array}{c} \mathcal{B}\omega \\ \mathcal{H}^+ \end{array} \begin{array}{c} \not\supseteq \\ \subsetneq \end{array} \mathcal{H}^*.$$

In the seventies, Sallé was working with Coppo and Dezani on type systems for studying termination properties of λ-terms [26, 7]. In 1979, at the conference on λ-calculus that took place in Swansea, he conjectured that a strict inclusion $\mathcal{B}\omega \subsetneq \mathcal{H}^+$ holds. Such a conjecture was reported in the proof of [2, Thm. 17.4.16], but for almost fourty years no progress has been made in that direction. In 2016, the second and third authors with Breuvart and Ruoppolo proved that $\mathcal{H}^+$ satisfies the ω-rule [5], so $\mathcal{B}\omega \subseteq \mathcal{H}^+$. In this paper we demonstrate that the λ-theories $\mathcal{B}\omega$ and $\mathcal{H}^+$ actually coincide, thus disproving Sallé's conjecture.

To prove such a result we need to show that, whenever two λ-terms M and N are equal in $\mathcal{H}^+$, they are also equal in $\mathcal{B}\omega$. From [12], we know that in this case there is a Böhm tree T such that $\mathrm{BT}(M) \leq^\eta T \geq^\eta \mathrm{BT}(N)$, where $T' \leq^\eta T$ means that the Böhm tree T can be obtained from T' by performing countably many finite η-expansions. Thus, the Böhm trees of M, N are compatible and have a common "η-supremum" T.

Our proof can be divided into several steps:

1. We show that the aforementioned η-supremum T is *λ-definable*: there exists a λ-term P such that $\mathrm{BT}(P) = T$ (Proposition 39).
2. We apply the ω-rule to equate the Böhm tree of the *stream* (infinite list) $\langle\!\langle \eta \rangle\!\rangle$ containing

all finite η-expansions of the identity, and the Böhm tree of the stream $\langle\!\langle \mathtt{I} \rangle\!\rangle$ containing infinitely many copies of the identity (Corollary 42).

3. We define a λ-term Ξ (Definition 35) taking as arguments the *codes* $\lceil\cdot\rceil$ of two λ-terms M_1, M_2 and a stream S, and such that, whenever $\mathrm{BT}(M_1) \leq^{\eta} \mathrm{BT}(M_2)$ holds:
   (i) $\mathrm{BT}(\Xi\lceil M_1\rceil\lceil M_2\rceil\langle\!\langle \eta \rangle\!\rangle) = \mathrm{BT}(M_2)$ (Lemma 37),
   (ii) $\mathrm{BT}(\Xi\lceil M_1\rceil\lceil M_2\rceil\langle\!\langle \mathtt{I} \rangle\!\rangle) = \mathrm{BT}(M_1)$ (Lemma 38).
3. Summing up, if M, N are equal in $\mathcal{H}^+$, then by (1) there is a λ-term P such that $\mathrm{BT}(M) \leq^{\eta} \mathrm{BT}(P) \geq^{\eta} \mathrm{BT}(N)$. Since $\mathcal{B}\omega$ equates all λ-terms having the same Böhm tree, we obtain the following sequence of equalities:

$$M =_{3(ii)} \Xi\lceil M\rceil\lceil P\rceil\langle\!\langle \mathtt{I} \rangle\!\rangle =_{(2)} \Xi\lceil M\rceil\lceil P\rceil\langle\!\langle \eta \rangle\!\rangle =_{3(i)} P$$
$$N =_{3(ii)} \Xi\lceil N\rceil\lceil P\rceil\langle\!\langle \mathtt{I} \rangle\!\rangle =_{(2)} \Xi\lceil N\rceil\lceil P\rceil\langle\!\langle \eta \rangle\!\rangle =_{3(i)} P$$

so M and N are equal in $\mathcal{B}\omega$ (Theorem 43).

The intuition behind $\Xi\lceil M\rceil\lceil N\rceil S$ is that, working on their codes, the λ-term Ξ computes the Böhm trees of M and N, compares them, and at every position applies to the "smaller" (less η-expanded) an element extracted from the stream S in the attempt of matching the structure of the "larger". When S contains all possible η-expansions each attempt succeeds, so $\Xi\lceil M\rceil\lceil N\rceil\langle\!\langle \eta \rangle\!\rangle$ computes the η-supremum of $\mathrm{BT}(M)$ and $\mathrm{BT}(N)$. When S only contains the identity, each non-trivial attempt fails, and $\Xi\lceil M\rceil\lceil N\rceil\langle\!\langle \mathtt{I} \rangle\!\rangle$ computes their η-infimum.

We announce that the technique developed can be also used to prove that two λ-terms M and N are equal in $\mathcal{B}\eta$ exactly when their Böhm trees are equal up to countably many η-expansions of *bounded size*. This result is beyond the scope of the present paper and omitted, but confirms the informal intuition about $\mathcal{B}\eta$ discussed by Barendregt in [2, §16.4].

Discussion

We build on the characterizations of $\mathcal{H}^+$ and $\mathcal{H}^*$ given by Hyland and Wadsworth [12, 13, 30] and subsequently improved by Lévy [17]. In Section 3 we give a uniform presentation of these preliminary results using the formulation given in [2, §19.2] for $\mathcal{H}^*$, that exploits the notion of Böhm-like trees, namely labelled trees that "look like" Böhm trees but might not be λ-definable. Böhm-like trees were introduced in [2] since at the time researchers were less familiar with the notion of coinduction, but they actually correspond to infinitary terms coinductively generated by the grammar of normal λ-terms possibly containing the constant $\bot$. It is worth mentioning that such characterizations of $\mathcal{H}^+$ and $\mathcal{H}^*$ have been recently rewritten by Severi and de Vries using the modern approach of infinitary rewriting [28, 29], and that we could have used their formulation instead.

A key ingredient in our proof is the fact that λ-terms can be encoded with natural numbers, and therefore with Church numerals, in an effective way. This is related to the theory of self-interpreters in λ-calculi, which is an ongoing subject of study [21, 10, 24, 6], and we believe that the present paper provides a nice illustration of the usefulness of such interpreters. As a presentation choice, we decided to use the encoding described in Barendregt's book [2, Def.6.5.6], even if it works for closed λ-terms only, because it is the most standard. However, our construction could be recast using any (effective) encoding, like the one proposed by Mogensen in [21] that works more generally for open terms.

Outline

After the preliminary Section 2, we review the main notions of extensional equalities on Böhm trees in Section 3, and the key results concerning the ω-rule in Section 4. In Section 5

we show how to build Böhm trees, and their η-supremum, starting from codes of λ-terms and streams of η-expansions. Finally, Section 6 is devoted to the proof of $\mathcal{B}\omega = \mathcal{H}^+$.

2 Preliminaries

2.1 The Lambda Calculus

We generally use the notation of Barendregt's book [2] for λ-calculus. The set Λ of λ-terms over an infinite set Var of variables is defined by the following grammar:

$$\Lambda : \qquad M, N ::= \quad x \;\mid\; \lambda x.M \;\mid\; MN \qquad (\text{for } x \in \text{Var})$$

The application associates to the left and has a higher precedence than λ-abstraction. For instance $\lambda xyz.xyz = \lambda x.(\lambda y.(\lambda z.((xy)z)))$. We write $MN^{\sim n}$ for $MN\cdots N$ (n times).

The set $\mathsf{fv}(M)$ of *free variables* of M and the *α-conversion* are defined as in [2, Ch. 1§2]. Hereafter, we consider λ-terms up to α-conversion. A λ-term M is called *closed* whenever $\mathsf{fv}(M) = \emptyset$ and we denote by Λ^o the set of all closed λ-terms.

The *β-reduction* is defined as usual $(\lambda x.M)N \to_\beta M[N/x]$ where $M[N/x]$ denotes the capture-free substitution of N for all free occurrences of x in M. We denote by $\mathtt{nf}_\beta(M)$ the β-normal form of M, if it exists. The *η-reduction* is given by $\lambda x.Mx \to_\eta M$ subject to the usual proviso $x \notin \mathsf{fv}(M)$. Given $\to_{\mathtt{R}}$, write $=_{\mathtt{R}}$ ($\twoheadrightarrow_{\mathtt{R}}$) for $\mathtt{R}$-conversion (multistep $\mathtt{R}$-reduction).

We will use the following notations for specific λ-terms:

$$\mathtt{I} = \mathtt{1}^0 = \lambda x.x, \quad \mathtt{1}^{n+1} = \lambda xz.x(\mathtt{1}^n z), \quad \mathtt{B} = \lambda fgx.f(gx), \quad \mathtt{Y} = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)),$$
$$\mathtt{K} = \lambda xy.x, \qquad \mathtt{F} = \lambda xy.y, \qquad \Omega = (\lambda x.xx)(\lambda x.xx), \quad \mathtt{J} = \mathtt{Y}(\lambda jxz.x(jz)),$$

where $\mathtt{I}$ is the identity, $\mathtt{1}^n$ is a $\beta\eta$-expansion of $\mathtt{I}$, $\mathtt{B}$ is the composition combinator $M \circ N = \mathtt{B}MN$, $\mathtt{K}$ and $\mathtt{F}$ are the first and second projection, Ω is the paradigmatic looping λ-term, $\mathtt{Y}$ is Curry's fixed point and $\mathtt{J}$ is Wadsworth's combinator [30]. We denote by c_n the n-th Church numeral [2, Def. 6.4.4], by succ and pred the successor and predecessor, and by $\mathsf{ifz}(\mathrm{c}_n, M, N)$ the λ-term which is equal to M if $n = 0$ and is equal to N otherwise.

The *pairing* is encoded in λ-calculus by setting $[M, N] = \lambda y.yMN$ for $y \notin \mathsf{fv}(MN)$ [2, Def. 6.2.4].

▶ **Definition 1.** An enumeration of closed λ-terms $e = (M_0, M_1, M_2, \dots)$ is called *effective* (or *uniform* in [2, §8.2]) if there is $F \in \Lambda^o$ such that $F\mathrm{c}_n =_\beta M_n$.

Given an effective enumeration, we define (using $\mathtt{Y}$ like in [2, Def. 8.2.3]) the *sequence* $[M_n]_{n\in\mathbb{N}}$ as a single λ-term satisfying $[M_n]_{n\in\mathbb{N}} =_\beta [M_0, [M_{n+1}]_{n\in\mathbb{N}}]$. We often use the notations:

$$[M_n]_{n\in\mathbb{N}} = [M_0, [M_1, [M_2, \dots]]] = [M_0, M_1, M_2, \dots].$$

The *i-th projection* is $\pi_i = \lambda y.y\mathtt{F}^{\sim i}\mathtt{K}$ since $\pi_i[M_n]_{n\in\mathbb{N}} =_\beta M_i$.

▶ **Definition 2.** Starting from a sequence $S = [M_n]_{n\in\mathbb{N}}$ we can build a *stream* $\langle\!\langle S \rangle\!\rangle = \lambda\vec{x}.[M_n\vec{x}]_{n\in\mathbb{N}}$ having $\mathtt{P}_i = \lambda s\vec{x}.\pi_i(s\vec{x})$ as projection.

The difference between a sequence and a stream stands in their applicative behaviour: when applying $\langle\!\langle S \rangle\!\rangle$ to $\vec{P}$ all λ-terms in $\langle\!\langle S \rangle\!\rangle$ receive $\vec{P}$ as arguments, i.e., $\langle\!\langle S \rangle\!\rangle\vec{P} =_\beta [M_n\vec{P}]_{n\in\mathbb{N}}$.

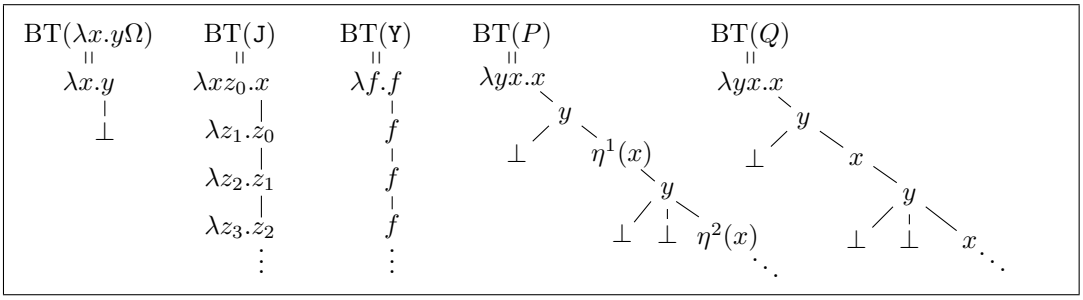


Figure 1 Some examples of Böhm trees, where $\eta^n(x) = \mathrm{BT}(\mathtt{1}^n x)$. We refer to [2, Lemma 16.4.4] for the definition of P, Q.

2.2 Solvability and Böhm(-like) Trees

The λ-terms are classified into solvable and unsolvable, depending on their capability of interaction with the environment.

▶ **Definition 3.** A closed λ-term N is *solvable* if there are $\vec{P} \in \Lambda$ such that $N\vec{P} \twoheadrightarrow_\beta \mathtt{I}$. A λ-term M is *solvable* if its closure $\lambda\vec{x}.M$ is solvable. Otherwise M is called *unsolvable*.

A λ-term M is in *head normal form* (*hnf*) if it has the shape $\lambda x_1 \dots x_n.x_j M_1 \cdots M_k$ where either $x_j \in \vec{x}$ or it is free. If M has an hnf, it can be reached by *head reductions* $\twoheadrightarrow_h$, i.e. by repeatedly contracting its *head redex* $\lambda\vec{x}.\underline{(\lambda y.P)Q}M_1 \cdots M_k$. As shown by Wadsworth in [30], a λ-term M is solvable if and only if M has a head normal form. The typical example of an unsolvable is Ω. Any $M \in \Lambda^o$ can be turned into an unsolvable by applying enough Ω's.

▶ **Lemma 4.** *[2, Lemma 17.4.4] For all $M \in \Lambda^o$ there is $k \in \mathbb{N}$ such that $M\Omega^{\sim k}$ is unsolvable.*

▶ **Definition 5.** The *Böhm tree* $\mathrm{BT}(M)$ of a λ-term M is defined coinductively as follows:

- if M is unsolvable then $\mathrm{BT}(M) = \bot$;
- if M is solvable and $M \twoheadrightarrow_h \lambda x_1 \dots x_n.x_j M_1 \cdots M_k$ then:

$$\mathrm{BT}(M) = \begin{array}{c} \lambda x_1 \dots x_n.x_j \\ \hline \mathrm{BT}(M_1) \quad \cdots \quad \mathrm{BT}(M_k) \end{array}$$

Notable examples of Böhm trees are given in Figure 1.

Some of the results that we use were originally formulated for "Böhm-like" trees, so we recall their definition [2, Def. 10.1.12].

▶ **Definition 6.** A *Böhm-like* tree T is a partial function $T : \mathbb{N}^* \to \mathcal{L} \times \mathbb{N}$, where $\mathbb{N}^*$ is the set of finite sequences of natural numbers and $\mathcal{L} = \{\lambda\vec{x}.y \mid \vec{x}, y \in \mathrm{Var}\}$, such that $\mathrm{dom}(T)$ is closed under prefixes and for all positions $\sigma \in \mathrm{dom}(T)$ and $n \in \mathbb{N}$ if their concatenation $\sigma.n$ belongs to $\mathrm{dom}(T)$ then $n < \pi_2(T(\sigma))$ holds. We denote by $\mathbb{BT}$ the set of all Böhm-like trees.

Intuitively, we have $T(\sigma) = (\lambda\vec{x}.y, n)$ if the node of T in position σ is labelled with "$\lambda\vec{x}.y$" and has n (possibly undefined) children. Given a Böhm-like tree T, its *underlying naked tree* $|T|$ is given by $\{\langle\rangle\} \cup \{\sigma.k \in \mathbb{N}^* \mid \pi_2(T(\sigma)) = n \text{ and } k < n\}$. The positions $\sigma \in |T| - \mathrm{dom}(T)$ correspond to unsolvable λ-terms, so we write $T(\sigma) = \bot$.

By [2, Thm. 10.1.23], $T \in \mathbb{BT}$ is partial computable and $\mathsf{fv}(T)$ is finite if and only if there is a λ-term M such that $\mathrm{BT}(M) = T$.

We will systematically confuse finite (resp. infinite) Böhm-like trees $T \in \mathbb{BT}$ with the corresponding (infinitary) λ-terms and use the same notations.

▶ **Lemma 7** (cf. [2, 10.1.5(v)]). *Let $M_i, N_i \in \Lambda$, for $i \in \mathbb{N}$. If for all $i \in \mathbb{N}$ we have* $\mathrm{BT}(M_i) = \mathrm{BT}(N_i)$ *then* $\mathrm{BT}([M_n]_{n\in\mathbb{N}}) = \mathrm{BT}([N_n]_{n\in\mathbb{N}})$.

2.3 Observational Equivalences and Lambda Theories

Observational equivalences and λ-theories become the main object of study when considering the computational equivalence more important than the process of computation.

▶ **Definition 8.** A *λ-theory* is a congruence on Λ (that is, an equivalence relation compatible with lambda abstraction and application) containing the β-conversion.

Given a λ-theory $\mathcal{T}$ we will write $\mathcal{T} \vdash M = N$, or simply $M =_{\mathcal{T}} N$, to express the fact that M and N are equal in $\mathcal{T}$. The set of all λ-theories, ordered by inclusion, forms quite a rich complete lattice, as shown by Lusin and Salibra in [18].

▶ **Definition 9.** A λ-theory $\mathcal{T}$ is called:
- *consistent* if it does not equate all λ-terms;
- *extensional* if it contains the η-conversion;
- *sensible* if it equates all unsolvable λ-terms.

We denote by $\boldsymbol{\lambda}$ the least λ-theory, by $\boldsymbol{\lambda}\eta$ the least extensional λ-theory, by $\mathcal{H}$ the least sensible λ-theory, by $\mathcal{B}$ the λ-theory equating all λ-terms having the same Böhm tree, and by $\mathcal{H}^*$ the (unique) greatest consistent sensible λ-theory.

The λ-theory $\mathcal{B}$ is sensible, thus we have $\boldsymbol{\lambda} \subsetneq \mathcal{H} \subsetneq \mathcal{B} \subsetneq \mathcal{H}^*$.

Given a λ-theory $\mathcal{T}$, we write $\mathcal{T}\eta$ for the least extensional λ-theory containing $\mathcal{T}$. Since $\mathcal{T} \subseteq \mathcal{T}'$ entails $\mathcal{T}\eta \subseteq \mathcal{T}'\eta$, we also have $\boldsymbol{\lambda}\eta \subseteq \mathcal{H}\eta \subseteq \mathcal{B}\eta \subseteq \mathcal{H}^*$ and actually all these inclusions turn out to be strict [2, Thm. 17.4.16].

▶ Remark 10. It is well known (see [2, Rem. 4.1.2]) that two λ-theories $\mathcal{T}, \mathcal{T}'$ that coincide on closed terms must be equal, hence we often focus on closed λ-terms.

Several interesting λ-theories are obtained via observational equivalences defined with respect to a set $\mathcal{O}$ of *observables*. Recall that a *context* $C[]$ is a λ-term with a *hole* denoted by $[]$. We write $C[M]$ for the λ-term obtained from $C[]$ by substituting M for the hole, possibly with capture of free variables in M.

Given $\mathcal{O} \subseteq \Lambda$, we write $M \in_\beta \mathcal{O}$ for $M \twoheadrightarrow_\beta M' \in \mathcal{O}$.

▶ **Definition 11.** Given a set $\mathcal{O} \subseteq \Lambda$, the *$\mathcal{O}$-observational equivalence* $\equiv^{\mathcal{O}}$ is defined by setting:

$$M \equiv^{\mathcal{O}} N \text{ if and only if } \forall C[](C[M] \in_\beta \mathcal{O} \iff C[N] \in_\beta \mathcal{O}).$$

We mainly focus on the following observational equivalences:
- Hyland/Wadsworth's observational equivalence $\equiv^{\mathtt{hnf}}$ is obtained by taking as $\mathcal{O}$ the set of head normal forms [13, 30].
- Morris's equivalence $\equiv^{\mathtt{nf}}$ is generated by taking as $\mathcal{O}$ the set of β-normal forms [22].

We will now see that $\equiv^{\mathtt{nf}}$ and $\equiv^{\mathtt{hnf}}$ have been characterized in terms of extensional equalities between Böhm trees.

3 Böhm Trees and Extensionality

We review three different notions of extensional equality between Böhm trees corresponding to the equality in $\mathcal{B}\eta, \mathcal{H}^+$ and $\mathcal{H}^*$. We start by analyzing the η-expansions of the identity.

3.1 The η-Expansions of The Identity

Let $\mathcal{I}^\eta \subseteq \Lambda$ be the set of finite η-expansions of the identity, that is $Q \in \mathcal{I}^\eta$ whenever $Q \twoheadrightarrow_{\beta\eta} \mathtt{I}$. The structural properties of such η-expansions have been analyzed in [14] (where this more liberal terminology is introduced). For instance, it is proved that $(\mathcal{I}^\eta, \circ, \mathtt{I})$ is an idempotent commutative monoid which is moreover closed under λ-calculus application.

▶ **Lemma 12.** *For $Q \in \Lambda$, the following are equivalent:*
(i) *$Q \in \mathcal{I}^\eta$, i.e. $Q \twoheadrightarrow_{\beta\eta} \mathtt{I}$,*
(ii) *$Q =_\beta \lambda y z_1 \dots z_m.yQ_1 \cdots Q_m$ such that $\lambda z_\ell.Q_\ell \in \mathcal{I}^\eta$,*
(iii) *$Q =_\beta \lambda y.Q'$ such that $Q' \twoheadrightarrow_{\beta\eta} y$.*

There is a one-to-one correspondence between elements of $\mathcal{I}^\eta$ in β-normal forms and finite (unlabelled) trees [14]. Clearly, $\mathtt{1}^n \in \mathcal{I}^\eta$, every $Q \in \mathcal{I}^\eta$ is β-normalizing, $\mathtt{nf}_\beta(Q)$ is a closed λ-term and $\mathrm{BT}(Q)$ is finite and does not contain any occurrence of $\bot$.

▶ **Definition 13.** Given $Q \in \mathcal{I}^\eta$ its *depth* (resp. *branching number*) is the height (resp. maximal number of branching) of its Böhm tree. The *size* of Q is the maximum between its depth and its branching number.

There are also λ-terms, like Wadsworth's $\mathtt{J}$, that *look* like η-expansions of $\mathtt{I}$ but give rise to infinite computations:

$$\mathtt{J} =_\beta \lambda x z_0.x(\mathtt{J}z_0) =_\beta \lambda x z_0.x(\lambda z_1.z_0(\mathtt{J}z_1)) =_\beta \cdots$$

The Böhm tree of $\mathtt{J}$ is an *infinite η-expansion* of the identity, a notion that is discussed in Section 3.4.

3.2 $\mathcal{B}\eta$: Countably Many η-Expansions of Bounded Size

Recall that $\mathcal{B}\eta$ is the least extensional λ-theory including $\mathcal{B}$. One might think that if $M =_{\mathcal{B}\eta} N$ then $\mathrm{BT}(M)$ and $\mathrm{BT}(N)$ differ because of finitely many η-expansions. In reality, one η-expansion of M can generate countably many η-expansions in its Böhm tree.

Consider, for instance, the following streams:

$$\langle\!\langle \mathtt{I} \rangle\!\rangle x = [x, x, x, \dots], \qquad \langle\!\langle \mathtt{1} \rangle\!\rangle x = [\mathtt{1}x, \mathtt{1}x, \mathtt{1}x, \dots], \qquad \langle\!\langle \mathtt{1}^* \rangle\!\rangle x = [\mathtt{1}^1 x, \mathtt{1}^2 x, \mathtt{1}^3 x, \dots].$$

whose Böhm trees are depicted in Figure 2. We have that $\langle\!\langle \mathtt{1} \rangle\!\rangle =_{\mathcal{B}} \mathtt{Y}(\lambda m x.[\lambda z.xz, mx]) \to_\eta \mathtt{Y}(\lambda m x.[x, mx]) =_{\mathcal{B}} \langle\!\langle \mathtt{I} \rangle\!\rangle$ thus $\langle\!\langle \mathtt{I} \rangle\!\rangle$ and $\langle\!\langle \mathtt{1} \rangle\!\rangle$ are equated in $\mathcal{B}\eta$ despite the fact that their Böhm trees differ by infinitely many η-expansions. More precisely, $M \to_\eta N$ entails that $\mathrm{BT}(M)$ can be obtained from $\mathrm{BT}(N)$ by performing at most *one* η-expansion at every position (see [2, Lemma 16.4.3]). The proof technique that we develop in Section 5 allows to demonstrate that two λ-terms M and N are equated in $\mathcal{B}\eta$ exactly when their Böhm trees are equal up to countably many η-expansions whose sizes are bound by some natural number k.

In particular, no finite amount of η-expansions in $\langle\!\langle \mathtt{I} \rangle\!\rangle$ can turn its Böhm tree into $\mathrm{BT}(\langle\!\langle \mathtt{1}^* \rangle\!\rangle)$, which has infinitely many η-expansions *of increasing depth*.

▶ **Corollary 14.** *$\mathcal{B}\eta \vdash \langle\!\langle \mathtt{I} \rangle\!\rangle = \langle\!\langle \mathtt{1} \rangle\!\rangle$, while $\mathcal{B}\eta \vdash \langle\!\langle \mathtt{I} \rangle\!\rangle \neq \langle\!\langle \mathtt{1}^* \rangle\!\rangle$.*

3.3 $\mathcal{H}^+$: Countably Many Finite η-Expansions

Let $\mathcal{H}^+$ be the λ-theory corresponding to Morris's original observational equivalence $\equiv^{\mathtt{nf}}$ where the observables are the β-normal forms [22]. The λ-theory $\mathcal{H}^+$ has been studied both

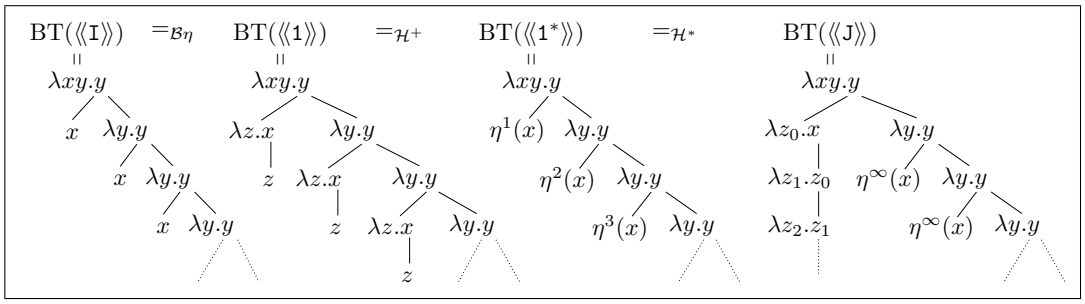


Figure 2 The Böhm trees of $\langle\!\langle \mathtt{I} \rangle\!\rangle, \langle\!\langle \mathtt{1} \rangle\!\rangle, \langle\!\langle \mathtt{1}^* \rangle\!\rangle$ and $\langle\!\langle \mathtt{J} \rangle\!\rangle$, where we set $\eta^n(x) = \mathrm{BT}(\mathtt{1}^n x)$ and $\eta^\infty(x) = \mathrm{BT}(\mathtt{J}x)$.

from a syntactic and semantic point of view in [8, 17, 5]. (The properties we present here can be found in [25, §11.2].) Two λ-terms having the same Böhm tree cannot be distinguished by any context $C[]$, so we have $\mathcal{B} \subseteq \mathcal{H}^+$. Since the η-reduction is strongly normalizable, a λ-term M has a β-normal form exactly when it has a $\beta\eta$-normal form, hence $\mathcal{H}^+$ is an extensional λ-theory. Therefore we have $\mathcal{B}\eta \subseteq \mathcal{H}^+$.

The question naturally arising is whether there are λ-terms different in $\mathcal{B}\eta$ that become equal in $\mathcal{H}^+$. It turns out that $\mathcal{H}^+ \vdash M = N$ holds exactly when $\mathrm{BT}(M)$ and $\mathrm{BT}(N)$ are equal up to countably many η-expansions of finite depth. A typical example of this situation is given by $\langle\!\langle \mathtt{I} \rangle\!\rangle$ and $\langle\!\langle \mathtt{1}^* \rangle\!\rangle$. The next definition is coinductive on the Böhm-like trees.

▶ **Definition 15.** For all $T, T' \in \mathbb{BT}$, we have $T \leq^\eta T'$ if either $T = T' = \bot$, or $T = \lambda\vec{x}.x_j T_1 \cdots T_k$ and $T' = \lambda\vec{x} z_1 \ldots z_m.x_j T'_1 \cdots T'_k Q_1 \cdots Q_m$, for $\vec{T}, \vec{T'} \in \mathbb{BT}$ and β-normal $\vec{Q} \in \Lambda$ such that $T_i \leq^\eta T'_i$ for all $i \leq k$, $z_\ell \notin \mathsf{fv}(x_j \vec{T} \vec{T'})$ and $\lambda z_\ell.Q_\ell \in \mathcal{I}^\eta$ for all $\ell \leq m$.

It is easy to check that $\mathrm{BT}(\langle\!\langle \mathtt{I} \rangle\!\rangle) \leq^\eta \mathrm{BT}(\langle\!\langle \mathtt{1}^* \rangle\!\rangle)$ holds.

▶ **Definition 16.** For $M, N \in \Lambda$, we write $M \leq^\eta N$ if and only if $\mathrm{BT}(M) \leq^\eta \mathrm{BT}(N)$.

Note that $M \leq^\eta N$ and $N \leq^\eta M$ entail $\mathrm{BT}(M) = \mathrm{BT}(N)$, so the equivalence corresponding to $\leq^\eta$ and capturing $=_{\mathcal{H}^+}$ needs to be defined in the following more subtle way.

▶ **Theorem 17** (Hyland [12], see also [17])**.** *For all $M, N \in \Lambda$, $\mathcal{H}^+ \vdash M = N$ if and only if there is a Böhm-like tree $T \in \mathbb{BT}$ such that $\mathrm{BT}(M) \leq^\eta T \geq^\eta \mathrm{BT}(N)$.*

So, in general, when $M =_{\mathcal{H}^+} N$, one may need to perform countably many η-expansions both in $\mathrm{BT}(M)$ and in $\mathrm{BT}(N)$ to equate them and find the common "supremum".

▶ **Corollary 18.** $\mathcal{H}^+ \vdash \langle\!\langle \mathtt{I} \rangle\!\rangle = \langle\!\langle \mathtt{1}^* \rangle\!\rangle$, *while* $\mathcal{H}^+ \vdash \mathtt{I} \neq \mathtt{J}$.

3.4 $\mathcal{H}^*$: Countably Many Infinite η-Expansions

The theory $\mathcal{H}^*$ is, by far, the most well studied λ-theory. It corresponds to the observational equivalence $\equiv^{\mathtt{hnf}}$ where the observables are the head normal forms. It is also the maximal consistent sensible λ-theory [2, Thm. 16.2.6] and the theory of Scott's original λ-model $\mathcal{D}_\infty$ [27]. It is not difficult to check that $M \equiv^{\mathtt{nf}} N$ entails $M \equiv^{\mathtt{hnf}} N$, therefore $\mathcal{H}^+ \subseteq \mathcal{H}^*$.

Two λ-terms M, N are equated in $\mathcal{H}^*$ if their Böhm trees are equal up to countably many η-expansions of possibly infinite depth. The typical example is $\mathtt{I} =_{\mathcal{H}^*} \mathtt{J}$. However, $\mathtt{J}$ is not the only candidate: for every infinite (unlabelled) recursive tree T it is possible to define a λ-term $\mathtt{J}_T$ whose Böhm tree is an infinite η-expansion of the identity "following T" [5].

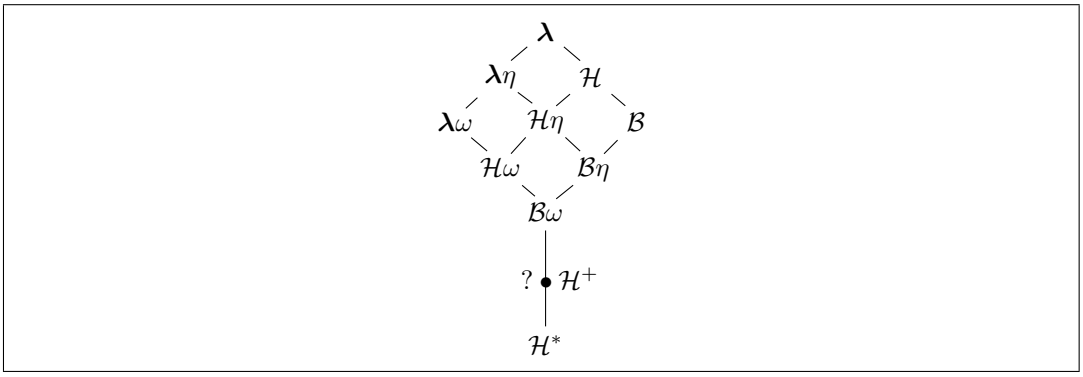


Figure 3 Barendregt's kite.

▶ **Definition 19.** For all $T, T' \in \mathbb{BT}$, we have $T \leq^{\eta}_{\omega} T'$ if either $T = T' = \bot$, or $T = \lambda\vec{x}.x_j T_1 \cdots T_k$ and $T' = \lambda\vec{x}z_1 \ldots z_m.x_j T'_1 \cdots T'_k T''_1 \cdots T''_m$, for $\vec{T}, \vec{T'}, \vec{T''} \in \mathbb{BT}$ such that $T_i \leq^{\eta}_{\omega} T'_i$ for all $i \leq k$, $z_\ell \notin \mathsf{fv}(x_j\vec{T}\vec{T'})$ and $z_\ell \leq^{\eta}_{\omega} T''_\ell$ for all $\ell \leq m$.

E.g., we have $\mathrm{BT}(\langle\!\langle \mathtt{I} \rangle\!\rangle) \leq^{\eta}_{\omega} \mathrm{BT}(\langle\!\langle \mathtt{J} \rangle\!\rangle)$ where $\langle\!\langle \mathtt{J} \rangle\!\rangle$ is defined by $\langle\!\langle \mathtt{J} \rangle\!\rangle x = [\mathtt{J}x, \mathtt{J}x, \mathtt{J}x, \ldots]$.

▶ **Theorem 20** (Hyland [13]/Wadsworth [30])**.** *For all $M, N \in \Lambda$, $\mathcal{H}^* \vdash M = N$ if and only if there is a Böhm-like tree $T \in \mathbb{BT}$ such that $\mathrm{BT}(M) \leq^{\eta}_{\omega} T \geq^{\eta}_{\omega} \mathrm{BT}(N)$.*

By Exercise 10.6.7 in [2], T can be always chosen to be the Böhm tree of some λ-term. As we will prove in Section 6, this property also holds for the tree T of Theorem 17.

▶ **Corollary 21.** *The streams $\langle\!\langle \mathtt{I} \rangle\!\rangle, \langle\!\langle \mathtt{1} \rangle\!\rangle, \langle\!\langle \mathtt{1}^* \rangle\!\rangle$ and $\langle\!\langle \mathtt{J} \rangle\!\rangle$ are all equal in $\mathcal{H}^*$. On the contrary, $\mathcal{B}\eta \vdash \langle\!\langle \mathtt{I} \rangle\!\rangle \neq \langle\!\langle \mathtt{1}^* \rangle\!\rangle$ and $\mathcal{H}^+ \vdash \langle\!\langle \mathtt{1}^* \rangle\!\rangle \neq \langle\!\langle \mathtt{J} \rangle\!\rangle$, so we have $\mathcal{B}\eta \subsetneq \mathcal{H}^+ \subsetneq \mathcal{H}^*$.*

4 The Omega Rule and Sallé's Conjecture

The ω-rule is a strong form of extensionality defined by:

$$(\omega) \qquad \forall P \in \Lambda^o.MP = NP \text{ entails } M = N.$$

Given a λ-theory $\mathcal{T}$ we denote its closure under the ω-rule by $\mathcal{T}\omega$. We say that $\mathcal{T}$ *satisfies the ω-rule*, written $\mathcal{T} \vdash \omega$, if $\mathcal{T} = \mathcal{T}\omega$. The ω-rule, and the question of which λ-theories satisfy it, has been extensively investigated by many authors [16, 1, 23, 3, 15].

The following lemma collects some results in [2, §4.1].

▶ **Lemma 22.** *For all λ-theories $\mathcal{T}$, we have:*
(i) *$\mathcal{T}\eta \subseteq \mathcal{T}\omega$,*
(ii) *$\rfloor\, \mathcal{T} \subseteq \mathcal{T}'$ entails $\mathcal{T}\omega \subseteq \mathcal{T}'\omega$.*

In general, because of the quantification over all $P \in \Lambda^o$, it can be difficult to understand what λ-terms different in $\mathcal{T}$ become equal in $\mathcal{T}\omega$, especially when $\mathcal{T}$ is extensional.

The picture in Figure 3, where $\mathcal{T}$ is above $\mathcal{T}'$ if $\mathcal{T} \subsetneq \mathcal{T}'$, is taken from [2, Thm. 17.4.16] and shows some facts about the λ-theories presented in Section 2.3 and the ω-rule.

The counterexample showing that $\boldsymbol{\lambda\eta} \nvdash \omega$ is based on complicated universal generators known as *Plotkin's terms* [2, Def. 17.3.26]. However, since these terms are unsolvable, they become useless when considering sensible λ-theories. We refer to [2, §17.4] for the proof of $\mathcal{H}\eta \nvdash \omega$ and rather discuss the validity of the ω-rule for λ-theories containing $\mathcal{B}$.

Let us consider two λ-terms P and Q whose Böhm trees are depicted in Figure 1. The Böhm trees of P and Q differ because of countably many finite η-expansions of increasing depth, therefore they are different in $\mathcal{B}\eta$ but equal in $\mathcal{H}^+$. This situation is analogous to what happens with $\langle\langle \mathtt{I} \rangle\rangle$ and $\langle\langle \mathtt{1}^* \rangle\rangle$, indeed $P \leq^\eta Q$ holds. Perhaps surprisingly, P and Q can also be used to prove that $\mathcal{B}\eta \subsetneq \mathcal{B}\omega$ since $\mathcal{B}\omega \vdash P = Q$ holds. Indeed, by Lemma 4, for every $M \in \Lambda^o$, there exists k such that $M\Omega^k$ becomes unsolvable. By inspecting Figure 1, we notice that in $\mathrm{BT}(P)$ the variable y is applied to an increasing number of Ω's (represented in the tree by $\bot$). So, when substituting some $M \in \Lambda^o$ for y in $\mathrm{BT}(Py)$, there will be a level k of the tree where $M\Omega\cdots\Omega$ becomes $\bot$, thus cutting $\mathrm{BT}(PM)$ at level k. The same reasoning can be done for $\mathrm{BT}(QM)$. Therefore $\mathrm{BT}(PM)$ and $\mathrm{BT}(QM)$ only differ because of finitely many η-expansions. Since $\mathcal{B}\eta \subseteq \mathcal{B}\omega$, we conclude that $PM =_{\mathcal{B}\omega} QM$ and therefore by the ω-rule $P =_{\mathcal{B}\omega} Q$. Such an argument is due to Barendregt [2, Lemma 16.4.4].

The fact that $\mathcal{H}^* \vdash \omega$ is an easy consequence of its maximality. However, there are several direct proofs: see [2, §17.2] for a syntactic demonstration and [30] for a semantic one.

A natural question, raised by Barendregt in [2, Thm. 17.4.16], concerns the position of $\mathcal{H}^+$ in the picture of Figure 3. In the proof of that theorem, it is mentioned that Sallé formulated the following conjecture (represented in the diagram with a question mark).

▶ **Conjecture 23.** $\mathcal{B}\omega \subsetneq \mathcal{H}^+$.

The longstanding open question whether $\mathcal{H}^+ \vdash \omega$ has been recently answered positively by Breuvart *et al.* in [5].

▶ **Theorem 24** ([5, Thm. 40])**.** $\mathcal{H}^+ \vdash \omega$.

From this it follows that $\mathcal{B}\omega \subseteq \mathcal{H}^+$. In Theorem 43 we show that $\mathcal{B}\omega = \mathcal{H}^+$, thus disproving Sallé's conjecture.

5 Building Böhm Trees by Codes and Streams

The key step for proving $\mathcal{H}^+ = \mathcal{B}\omega$ is to show that the tree T of Theorem 17 giving the "η-supremum" of M, N can be chosen to be the Böhm tree of a λ-term P (Proposition 39). Intuitively, the λ-term P will inspect the structure of M, N looking at their codes and choose the correct η-expansion to apply from a suitable stream. We start by showing that the Böhm tree of a λ-term can be reconstructed from its code.

5.1 Building Böhm Trees by Codes

Let $\# : \Lambda \to \mathbb{N}$ be an effective one-to-one map, associating with every λ-term M its *code* $\#M$ (the Gödel number of M). The *quote* $\ulcorner M \urcorner$ of M is the corresponding numeral $\mathrm{c}_{\#M}$. We now recall some well established facts from [2, §8.1]. By [2, Thm. 8.1.6], there is a λ-term $\mathtt{E} \in \Lambda^o$ such that $\mathtt{E}\ulcorner M \urcorner = M$ for all $M \in \Lambda^o$. This is false in general for open λ-terms M.

▶ Remark 25. The following operations are effective:

- from $\#M$ compute $\#M'$ where $M \to_h M'$ (since the head-reduction is an effective reduction strategy),
- from $\#(\lambda\vec{x}.x_j M_1 \cdots M_k)$ compute $\#M_i$ for $i \leq k$,
- from $\#M$ compute $\#(\lambda x_1 \ldots x_n.M)$ for $x_i \in \mathrm{Var}$.

From Remark 25 and Church's thesis, the following term Φ exists and can be defined using the fixed point combinator $\mathtt{Y}$.

▶ **Definition 26.** Let $\Phi \in \Lambda^o$ be such that for all $M \in \Lambda^o$:

- $\Phi\lceil M\rceil = \lambda \underline{x}_1 \dots \underline{x}_n.\underline{x}_j(L_1\underline{x}_1 \cdots \underline{x}_n) \cdots (L_k\underline{x}_1 \cdots \underline{x}_n)$ where $L_i = \Phi\lceil \lambda\vec{x}.M_i\rceil$ if $M \twoheadrightarrow_h \lambda x_1 \dots x_n.x_j M_1 \cdots M_k$.
- $\Phi\lceil M\rceil$ is unsolvable whenever M is unsolvable.

(The $\underline{x}_i$ are underlined to stress the fact that they are fresh.)

The term Φ builds the Böhm tree of M from its code $\#M$. Notice that the closure $\lambda\vec{x}.M_i$ on the recursive calls is needed to obtain a closed term (since $M \in \Lambda^o$ entails $\mathsf{fv}(M_i) \subseteq \vec{x}$). In the definition above we use the fact that $\mathcal{B}$ is a λ-theory, so $\mathrm{BT}(\lambda\vec{x}.M) = \lambda\vec{x}.\mathrm{BT}(M)$ thus the free variables $\underline{\vec{x}}$ can be reapplied externally. This commutation property between $\mathrm{BT}(-)$ and λ-abstraction will be silently used when proving statements about Böhm trees of closed λ-terms, as below.

▶ **Lemma 27.** *For all $M \in \Lambda^o$, $\mathcal{B} \vdash \Phi\lceil M\rceil = M$.*

Proof. If M is unsolvable then $\mathrm{BT}(\Phi\lceil M\rceil) = \mathrm{BT}(M) = \bot$. Otherwise M is solvable, so we have $M \twoheadrightarrow_h \lambda\vec{x}.x_j M_1 \cdots M_k$ and $\Phi\lceil M\rceil = \lambda\vec{x}.x_j(L_1\vec{x}) \cdots (L_k\vec{x})$ for $L_i = \Phi\lceil\lambda\vec{x}.M_i\rceil$. We conclude since by coinductive hypothesis $\mathrm{BT}(\Phi\lceil\lambda\vec{x}.M_i\rceil)\vec{x} = (\lambda\vec{x}.\mathrm{BT}(M_i))\vec{x} = \mathrm{BT}(M_i)$. ◀

5.2 η-Expanding Böhm Trees from Streams

The construction of Φ might look unimpressive in the sense that also the enumerator $\mathtt{E}$ enjoys the property $\mathrm{BT}(\mathtt{E}\lceil M\rceil) = \mathrm{BT}(M)$ for all $M \in \Lambda^o$. However, $\mathtt{E}$ does not satisfy the recursive equation of Definition 26, which has the advantage of exposing the structure of the tree and, doing so, opens the way for altering the tree. For instance, it is possible to modify Definition 26 in order to obtain a λ-term Ψ which builds an η-expansion of $\mathrm{BT}(M)$ starting from the code of M and a stream of η-expansions of the identity (cf. Section 3.1).

▶ **Definition 28.** Let $\vec{\eta} = (\eta_0, \eta_1, \eta_2, \dots)$ be an effective enumeration of all closed η-expansions of $\mathtt{I}$, i.e., of the set $\mathcal{I}^\eta \cap \Lambda^o$. Define the corresponding stream $\langle\!\langle\eta\rangle\!\rangle x = [\eta_0 x, \eta_1 x, \eta_2 x, \dots]$.

From now on, we fix the enumeration $\vec{\eta}$ and the stream $\langle\!\langle\eta\rangle\!\rangle$.

In order to decide what η-expansion is applied at a certain position σ in $\mathrm{BT}(M)$, we use a function $f(\sigma) = n$ and extract the η-expansion of index n from $\langle\!\langle\eta\rangle\!\rangle$. Since f needs to be computable, we fix an effective encoding $\# : \mathbb{N}^* \to \mathbb{N}$ of all finite sequences and consider f computable "after coding".

Notice that, since $\#$ is effective, from the code $\#\sigma$ it is possible to compute the code $\#(\sigma.i)$ for all $i \in \mathbb{N}$, and *vice versa*. We denote by $\lceil\sigma\rceil$ the corresponding numeral $\mathrm{c}_{\#\sigma}$.

In the following definition s is an arbitrary variable, but in practice we will always apply $\Psi_f\lceil M\rceil\lceil\sigma\rceil$ to some stream.

▶ **Definition 29.** Let $f : \mathbb{N}^* \to \mathbb{N}$ be a computable function, and $\Psi_f \in \Lambda^o$ be such that for all $M \in \Lambda^o$ and for all positions $\sigma \in \mathbb{N}^*$:

- $\Psi_f\lceil M\rceil\lceil\sigma\rceil s = \lambda\underline{x}_1 \dots \underline{x}_n.\mathtt{P}_{f(\sigma)}s(\underline{x}_j(L'_1\underline{\vec{x}}) \cdots (L'_k\underline{\vec{x}}))$ where $L'_i = \Psi_f\lceil\lambda x_1 \dots x_n.M_i\rceil\lceil\sigma.i\rceil\, s$ if $M \twoheadrightarrow_h \lambda x_1 \dots x_n.x_j M_1 \cdots M_k$.
- $\Psi_f\lceil M\rceil$ is unsolvable whenever M is unsolvable.

(Recall that $\mathtt{P}_i$ denotes the i-th projection for streams and the $\underline{x}_i$'s are fresh variables.)

The actual existence of such a Ψ_f follows from Remark 25, the effectiveness of the encodings, the computability of f and Church's thesis. We now verify that the λ-term $\Psi_f\lceil M\rceil\lceil\sigma\rceil$ when applied to the stream $\langle\!\langle\eta\rangle\!\rangle$ actually computes an η-expansion of $\mathrm{BT}(M)$ in the sense of Definition 15.

▶ **Lemma 30.** *Let $f : \mathbb{N}^* \to \mathbb{N}$ be a computable function. For all $M \in \Lambda^o$ and $\sigma \in \mathbb{N}^*$, $M \leq^\eta \Psi_f\lceil M\rceil\lceil\sigma\rceil\langle\langle\eta\rangle\rangle$.*

Proof. If M is unsolvable, $\mathrm{BT}(M) = \mathrm{BT}(\Psi_f\lceil M\rceil\lceil\sigma\rceil) = \bot$.

Otherwise $M \twoheadrightarrow_h \lambda\vec{x}.x_jM_1\cdots M_k$. Thus, for $f(\sigma) = q$ and $\eta_q = \lambda yz_1\cdots z_m.yQ_1\cdots Q_m$ where $\lambda z_i.Q_i \in \mathcal{I}^\eta$ we have

$$\begin{aligned}\Psi_f\lceil M\rceil\lceil\sigma\rceil\langle\langle\eta\rangle\rangle &=_\beta \lambda\vec{x}.\mathtt{P}_q\langle\langle\eta\rangle\rangle(x_j(L'_1\vec{x})\cdots(L'_k\vec{x}))\\ &=_\beta \lambda\vec{x}.\eta_q(x_j(L'_1\vec{x})\cdots(L'_k\vec{x})) =_\beta \lambda\vec{x}\vec{z}.x_j(L'_1\vec{x})\cdots(L'_k\vec{x})Q_1\cdots Q_m\end{aligned}$$

for $L'_i = \Psi_f\lceil\lambda\vec{x}.M_i\rceil\lceil\sigma.i\rceil\langle\langle\eta\rangle\rangle$. We conclude because by coinductive hypothesis $\mathrm{BT}(M_i) \leq^\eta \mathrm{BT}(L'_i\vec{x})$. ◀

Since Ψ_f picks the η-expansion to apply from the input stream, we can retrieve the behaviour of Φ by applying $\langle\langle\mathtt{I}\rangle\rangle$.

▶ **Lemma 31.** *Let $f : \mathbb{N}^* \to \mathbb{N}$ be computable. For all $M \in \Lambda^o$ and $\sigma \in \mathbb{N}^*$, we have $\mathcal{B} \vdash M = \Psi_f\lceil M\rceil\lceil\sigma\rceil\langle\langle\mathtt{I}\rangle\rangle$.*

Proof Sketch. As in the proof of Lemma 30, using the fact that $\mathtt{P}_q\langle\langle\mathtt{I}\rangle\rangle = \mathtt{I}$ for all $q \in \mathbb{N}$. ◀

5.3 Building the η-Supremum

Using similar techniques, we define a λ-term Ξ that builds from the codes of M, N and the stream $\langle\langle\eta\rangle\rangle$ the (smallest) η-supremum satisfying $M \leq^\eta \Xi\lceil M\rceil\lceil N\rceil\langle\langle\eta\rangle\rangle \geq^\eta N$, if it exists, that is whenever M and N are compatible (Proposition 39). Intuitively, at every position σ, Ξ needs to compare the structure of M, N at σ and apply the correct η_i taken from $\langle\langle\eta\rangle\rangle$.

Rather than proving that there exists a computable function $f : \mathbb{N}^* \to \mathbb{N}$ associating to every σ the corresponding η_i (which can be tedious) we use the following property of $\vec{\eta} = (\eta_0, \eta_1, \ldots, \eta_i, \ldots)$: since every closed η-expansion $Q \in \mathcal{I}^\eta$ is β-normalizable and the enumeration $\vec{\eta}$ is effective, it is possible to decide starting from the code $\#Q$ the index i of Q in $\vec{\eta}$. Moreover, it is possible to choose such an i minimal.

▶ **Lemma 32.** *There exists a computable function $\iota : \mathbb{N} \to \mathbb{N}$ such that, for all $M \in \Lambda^o$, if $M =_\beta \eta_i$ and $M \neq_\beta \eta_k$ for all $k < i$ then $\iota(\#M) = i$.*

Proof. Let $\delta(m, n)$ be the partial computable map satisfying for all normalizing $M, N \in \Lambda^o$: $\delta(\#M, \#N) = 0$ if M and N have the same β-normal form; $\delta(\#M, \#N) = 1$ otherwise. Then ι can be defined by setting $\iota(n) := \mu k.\delta(\#(\pi_k\vec{\eta}), n) = 0$. ◀

From now on we consider fixed such a function ι, which depends on the enumeration $\vec{\eta}$ generating the stream $\langle\langle\eta\rangle\rangle$.

▶ **Definition 33.** For $M, N \in \Lambda$, we define:

- **(i)** $M \leq_h N$ whenever $M \twoheadrightarrow_h \lambda\vec{x}.x_jM_1\cdots M_k$ and $N \twoheadrightarrow_h \lambda\vec{x}z_1\ldots z_m.x_jN_1\cdots N_kQ_1\cdots Q_m$ with $\lambda z_\ell.Q_\ell \in \mathcal{I}^\eta$ for all $\ell \leq m$,
- **(ii)** $M \sim_h N$ if both $M \leq_h N$ and $N \leq_h M$ hold,
- **(iii)** $M <_h N$ if $M \leq_h N$ holds but $M \not\sim_h N$.

Whenever $M \leq_h N$ holds, we say that N *looks like an η-expansion of M*. This does not necessarily mean that it actually is: for instance $\lambda z.x\mathtt{F}z \leq_h \lambda z.x\mathtt{K}z$ since we do not require that $\mathtt{F} \leq_h \mathtt{K}$ holds, and $z \leq_h \lambda z.zz$ since we do not check that $z \notin \mathsf{fv}(\mathrm{BT}(x_j\vec{M}\vec{N}))$. Therefore, compared with $\leq^\eta$ of Definition 15, the relation $\leq_h$ is weaker since it lacks the coinductive calls and the occurrence check on z_ℓ. This is necessary to ensure the following semi-decidability property.

$$\Xi_\iota\lceil M\rceil\lceil N\rceil s = \begin{cases} \lambda \underline{x}_1 \dots \underline{x}_n.\mathtt{P}_q s(\underline{x}_j(\Upsilon_1 \underline{x}_1 \cdots \underline{x}_n)\cdots(\Upsilon_k \underline{x}_1 \cdots \underline{x}_n)) & \text{if } M \leq_h N, \text{ with} \\ \text{where} \begin{bmatrix} q = \iota(\#(\mathtt{nf}_\beta(\lambda y z_1 \dots z_m.y \underline{Q}_1 \cdots Q_m))) & M \twoheadrightarrow_h \lambda x_1 \dots x_n.x_j M_1 \cdots M_k \text{ and} \\ \Upsilon_i := \Xi_\iota\lceil \lambda x_1 \dots x_n.M_i\rceil\lceil \lambda x_1 \dots x_n.N_i\rceil s & N \twoheadrightarrow_h \lambda x_1 \dots x_n z_1 \dots z_m.x_j N_1 \cdots N_k Q_1 \cdots Q_m; \end{bmatrix} \\ \Xi_\iota\lceil N\rceil\lceil M\rceil s & \text{if } N <_h M; \\ \Omega & \text{otherwise.} \end{cases}$$

Figure 4 The λ-term $\Xi_\iota \in \Lambda^o$ satisfies in $\mathcal{H}$ the recursive equation above.

▶ Remark 34. The property $M \leq_h N$ can be semi-decided:
- first head-reduce in parallel M, N until they reach a hnf,
- if both reductions terminate, compare the two hnfs and check whether they have the shape of Definition 33(i),
- then semi-decide whether $Q_\ell \twoheadrightarrow_{\beta\eta} z_\ell$ for all $\ell \leq m$.

This procedure might fail to terminate when $M \not\leq_h N$.

By Remarks 25 and 34, the fact that ι is computable (Lemma 32) and Church's thesis, the λ-term Ξ_ι below exists.

▶ **Definition 35.** Let $\iota : \mathbb{N} \to \mathbb{N}$ be the computable function of Lemma 32. We define $\Xi_\iota \in \Lambda^o$ such that for all $M, N \in \Lambda^o$ the recursive equation of Figure 4 is satisfied in $\mathcal{H}$.

There are some subtleties to discuss in the definition of Ξ_ι. The fact that $Q_\ell \twoheadrightarrow_{\beta\eta} z_\ell$ for all $\ell \leq m$, although not explicitly written, is a consequence of $M \leq_h N$. *A priori* $\lambda z_\ell.Q_\ell \in \mathcal{I}^\eta$ might be open (consider for instance $\lambda z_\ell.\mathtt{K} z_\ell y \to_\beta \mathtt{I}$) but its β-normal form is always a closed λ-term. This is the reason why we compute $\mathtt{nf}_\beta(\lambda y \vec{z}.y\vec{Q})$ before applying ι to its code. In particular, ι is defined on all the codes $\#(\mathtt{nf}_\beta(\lambda y \vec{z}.y\vec{Q}))$.

The following commutativity property follows from the second condition of Figure 4 and should be natural considering that $\Xi_\iota\lceil M\rceil\lceil N\rceil\langle\langle\eta\rangle\rangle$ is supposed to compute the η-join of $\mathrm{BT}(M)$ and $\mathrm{BT}(N)$ which is a commutative operation.

▶ **Lemma 36.** *For all $M, N \in \Lambda^o$, we have $\mathcal{B} \vdash \Xi_\iota\lceil M\rceil\lceil N\rceil = \Xi_\iota\lceil N\rceil\lceil M\rceil$.*

Proof. We proceed by coinduction on their Böhm trees.

If M, N are unsolvable or neither $M \leq_h N$ nor $N \leq_h M$ holds, then $\mathrm{BT}(\Xi_\iota\lceil M\rceil\lceil N\rceil) = \mathrm{BT}(\Xi_\iota\lceil N\rceil\lceil M\rceil) = \bot$. The cases $M <_h N$ and $N <_h M$ follow by definition.

If $M \sim_h N$, then we have $M \twoheadrightarrow_h \lambda\vec{x}.x_j M_1 \cdots M_k$ and $N \twoheadrightarrow_h \lambda\vec{x}.x_j N_1 \cdots N_k$. Since $\mathtt{P}_q s \twoheadrightarrow_\beta \lambda y.sy\mathtt{F}^{\sim q}\mathtt{K}$ we have

$$\Xi_\iota\lceil M\rceil\lceil N\rceil s =_\beta \lambda\vec{x}.(\lambda y.sy\mathtt{F}^{\sim q}\mathtt{K})(x_j(\Upsilon_1\vec{x})\cdots(\Upsilon_k\vec{x})) =_\beta \lambda\vec{x}.s(x_j(\Upsilon_1\vec{x})\cdots(\Upsilon_k\vec{x}))\mathtt{F}^{\sim q}\mathtt{K}$$
$$\Xi_\iota\lceil N\rceil\lceil M\rceil s =_\beta \lambda\vec{x}.(\lambda y.sy\mathtt{F}^{\sim q}\mathtt{K})(x_j(\Upsilon'_1\vec{x})\cdots(\Upsilon'_k\vec{x})) =_\beta \lambda\vec{x}.s(x_j(\Upsilon'_1\vec{x})\cdots(\Upsilon'_k\vec{x}))\mathtt{F}^{\sim q}\mathtt{K}$$

where, for all $i \leq k$, we have $\Upsilon_i = \Xi_\iota\lceil\lambda\vec{x}.M_i\rceil\lceil\lambda\vec{x}.N_i\rceil s$ and $\Upsilon'_i = \Xi_\iota\lceil\lambda\vec{x}.N_i\rceil\lceil\lambda\vec{x}.M_i\rceil s$. We conclude since, by coinductive hypothesis, we have $\mathrm{BT}(\Upsilon_i\vec{x}) = \mathrm{BT}(\Upsilon'_i\vec{x})$ for all $i \leq k$. ◀

Another property that we expect is that whenever $M \leq^\eta N$ the λ-term $\Xi_\iota\lceil M\rceil\lceil N\rceil\langle\langle\eta\rangle\rangle$ computes the Böhm tree of N.

▶ **Lemma 37.** *For all $M, N \in \Lambda^o$, if $M \leq^\eta N$ then $\mathcal{B} \vdash \Xi_\iota\lceil M\rceil\lceil N\rceil\langle\langle\eta\rangle\rangle = N$.*

Proof. By coinduction on their Böhm trees. If M, N are unsolvable then so is $\Xi_\iota\lceil M\rceil\lceil N\rceil$. Otherwise $M \leq^\eta N$ implies $M \twoheadrightarrow_h \lambda\vec{x}.x_j M_1 \cdots M_k$, $N \twoheadrightarrow_h \lambda\vec{x}z_1 \dots z_m.x_j N_1 \cdots N_k Q_1 \cdots Q_m$ where each $z_\ell \notin \mathsf{fv}(\mathrm{BT}(x_j\vec{M}\vec{N}))$, $\lambda z_\ell.Q_\ell \in \mathcal{I}^\eta$ and $M_i \leq^\eta N_i$. In particular $M \leq_h N$ holds, so the first condition of Figure 4 applies.

From $\lambda z_\ell.Q_\ell \in \mathcal{I}^\eta$ it follows that $\lambda y\vec{z}.y\vec{Q} \in \mathcal{I}^\eta$, therefore $\iota(\#\mathtt{nf}_\beta(\lambda y\vec{z}.y\vec{Q})) = q$ for some index q. Setting $\Upsilon_i = \Xi_\iota\lceil\lambda\vec{x}.M\rceil\lceil\lambda\vec{x}.N\rceil\langle\!\langle\eta\rangle\!\rangle$, easy calculations give:

$$\begin{aligned}\Xi_\iota\lceil M\rceil\lceil N\rceil\langle\!\langle\eta\rangle\!\rangle &=_\beta \lambda\vec{x}.\mathtt{P}_q\langle\!\langle\eta\rangle\!\rangle(x_j(\Upsilon_1\vec{x})\cdots(\Upsilon_k\vec{x}))\\ &=_\beta \lambda\vec{x}.(\lambda y\vec{z}.y\vec{Q})(x_j(\Upsilon_1\vec{x})\cdots(\Upsilon_k\vec{x})) =_\beta \lambda\vec{x}\vec{z}.x_j(\Upsilon_1\vec{x})\cdots(\Upsilon_k\vec{x})Q_1\cdots Q_m\end{aligned}$$

We conclude as, by coinductive hypothesis, we have $\mathrm{BT}(\Upsilon_i) = \mathrm{BT}(\lambda\vec{x}.N_i)$ for all $i \le k$. ◀

Under the assumption $M \le^\eta N$ we can also use Ξ_ι to retrieve the Böhm tree of M by applying the stream $\langle\!\langle\mathtt{I}\rangle\!\rangle$. Since ι has been defined as depending on the enumeration $\vec{\eta}$, $\iota(\#Q)$ still provides an index q such that $\mathtt{P}_q\langle\!\langle\eta\rangle\!\rangle = Q$ but when applied to $\langle\!\langle\mathtt{I}\rangle\!\rangle$ it necessarily gives $\mathtt{P}_q\langle\!\langle\eta\rangle\!\rangle = \mathtt{I}$. This technique is analogous to the one used in Lemma 31.

▶ **Lemma 38.** *For all $M, N \in \Lambda^o$, if $M \le^\eta N$ then $\mathcal{B} \vdash \Xi_\iota\lceil M\rceil\lceil N\rceil\langle\!\langle\mathtt{I}\rangle\!\rangle = M$.*

Proof. We proceed by coinduction on their Böhm trees. If M, N are both unsolvable, then also $\Xi_\iota\lceil M\rceil\lceil N\rceil$ must be. Otherwise $M \le^\eta N$ implies $M \twoheadrightarrow_h \lambda\vec{x}.x_jM_1\cdots M_k$ and $N \twoheadrightarrow_h \lambda\vec{x}z_1\ldots z_m.x_jN_1\cdots N_kQ_1\cdots Q_m$ where each $z_\ell \notin \mathsf{fv}(\mathrm{BT}(x_j\vec{M}\vec{N}))$, $\lambda z_\ell.Q_\ell \in \mathcal{I}^\eta$ and $M_i \le^\eta N_i$. In particular $M \le_h N$ holds, so the first condition of Figure 4 applies.

From $\lambda z_\ell.Q_\ell \in \mathcal{I}^\eta$ it follows that $\lambda y\vec{z}.y\vec{Q} \in \mathcal{I}^\eta$, therefore $\iota(\#\mathtt{nf}_\beta(\lambda y\vec{z}.y\vec{Q})) = q$ for some index q. Setting $\Upsilon_i = \Xi_\iota\lceil\lambda\vec{x}.M\rceil\lceil\lambda\vec{x}.N\rceil\langle\!\langle\mathtt{I}\rangle\!\rangle$, easy calculations give:

$$\begin{aligned}\Xi_\iota\lceil M\rceil\lceil N\rceil\langle\!\langle\mathtt{I}\rangle\!\rangle &=_\beta \lambda\vec{x}.\mathtt{P}_q\langle\!\langle\mathtt{I}\rangle\!\rangle(x_j(\Upsilon_1\vec{x})\cdots(\Upsilon_k\vec{x}))\\ &=_\beta \lambda\vec{x}.\mathtt{I}(x_j(\Upsilon_1\vec{x})\cdots(\Upsilon_k\vec{x})) =_\beta \lambda\vec{x}.x_j(\Upsilon_1\vec{x})\cdots(\Upsilon_k\vec{x})\end{aligned}$$

We conclude since, by coinductive hypothesis, we have $\mathrm{BT}(\Upsilon_i) = \mathrm{BT}(\lambda\vec{x}.M_i)$ for all $i \le k$. ◀

6 $\mathcal{B}\omega = \mathcal{H}^+$ and Sallé's Conjecture is False

This section is devoted to prove that $\mathcal{B}\omega = \mathcal{H}^+$ holds (Theorem 43). As mentioned earlier, the first step is to show that the term Ξ_ι defined in the previous section, when applied to $\mathcal{H}^+$-equivalent terms, actually computes their η-supremum.

▶ **Proposition 39.** *For all $M, N \in \Lambda^o$, $\mathcal{H}^+ \vdash M = N$ iff $M \le^\eta \Xi_\iota\lceil M\rceil\lceil N\rceil\langle\!\langle\eta\rangle\!\rangle \ge^\eta N$.*

Proof. ($\Leftarrow$) It follows directly from Theorem 17.

($\Rightarrow$) By Theorem 17, we known that there exists a Böhm-like tree $T \in \mathbb{BT}$ such that $\mathrm{BT}(M) \le^\eta T \ge^\eta \mathrm{BT}(N)$. As usual, we proceed by coinduction on the Böhm(-like) trees.

If M or N is unsolvable then $\mathrm{BT}(M) = \mathrm{BT}(N) = T = \bot$.

Otherwise, from $\mathrm{BT}(M) \le^\eta T \ge^\eta \mathrm{BT}(N)$ we have, say:

$$\begin{aligned}&M \twoheadrightarrow_h \lambda\vec{x}.x_jM_1\cdots M_k, \qquad N \twoheadrightarrow_h \lambda\vec{x}z_1\ldots z_m.x_jN_1\cdots N_kQ_1\cdots Q_m,\\ &\qquad T = \lambda\vec{x}z_1\ldots z_m\ldots z_{m'}.x_jT_1\cdots T_kQ'_1\cdots Q'_m\cdots Q'_{m'},\end{aligned}$$

such that $z_1, \ldots, z_{m'} \notin \mathsf{fv}(\mathrm{BT}(x_jM_1\cdots M_k)T_1\cdots T_k)$, $z_{m+1}, \ldots, z_{m'} \notin \mathsf{fv}(\mathrm{BT}(x_jN_1\cdots N_k))$, $\mathrm{BT}(M_i) \le^\eta T_i$ and $\mathrm{BT}(N_i) \le^\eta T_i$ for all $i \le k$, $Q_\ell \le^\eta Q'_\ell$ for all $\ell \le m$, and $\lambda z_{\ell'}.Q'_{\ell'} \in \mathcal{I}^\eta$ for all $\ell' > m$. By Lemma 12, $Q'_\ell \twoheadrightarrow_{\beta\eta} z_\ell$ so $Q_\ell \le^\eta Q'_\ell$ entails $\lambda z_\ell.Q_\ell \in \mathcal{I}^\eta$, hence $M \le_h N$. Setting $q = \iota(\#(\mathtt{nf}(\lambda xz_1\ldots z_m.xQ_1\cdots Q_m)))$ and $\Upsilon_i = \Xi_\iota\lceil\lambda\vec{x}.M\rceil\lceil\lambda\vec{x}.N\rceil\langle\!\langle\eta\rangle\!\rangle$, we obtain:

$$\begin{aligned}&\Xi_\iota\lceil M\rceil\lceil N\rceil\langle\!\langle\eta\rangle\!\rangle =_\beta \lambda\vec{x}.\mathtt{P}_q\langle\!\langle\eta\rangle\!\rangle(x_j(\Upsilon_1\vec{x})\cdots(\Upsilon_k\vec{x}))\\ &=_\beta \lambda\vec{x}.(\lambda xz.xQ_1\cdots Q_m)(x_j(\Upsilon_1\vec{x})\cdots(\Upsilon_k\vec{x})) =_\beta \lambda\vec{x}\vec{z}.x_j(\Upsilon_1\vec{x})\cdots(\Upsilon_k\vec{x})Q_1\cdots Q_m.\end{aligned}$$

This case follows from the coinductive hypotheses since, for all $i \le k$, $\lambda\vec{x}.M_i \le^\eta \Upsilon_i \ge^\eta \lambda\vec{x}.N_i$. The symmetric case $N <_h M$ is treated analogously, using Lemma 36. ◀

The second step towards the proof of Theorem 43 is to show that the streams $\langle\!\langle \mathtt{I} \rangle\!\rangle$ and $\langle\!\langle \eta \rangle\!\rangle$ are equated in $\mathcal{B}\omega$. To prove such a result, we are going to use the auxiliary streams:

- $\langle\!\langle \mathtt{I} \rangle\!\rangle^{\Omega} yx = [yx, y\Omega x, y\Omega^{\sim 2}x, y\Omega^{\sim 3}x, \dots]$,
- $\langle\!\langle \eta \rangle\!\rangle^{\Omega} yx = [y(\eta_0 x), y\Omega(\eta_1 x), y\Omega^{\sim 2}(\eta_2 x), y\Omega^{\sim 3}(\eta_3 x), \dots]$,

which are equal in $\mathcal{B}\omega$, for the same reason the λ-terms P, Q of Figure 1 are.

▶ **Lemma 40.** $\mathcal{B}\omega \vdash \langle\!\langle \mathtt{I} \rangle\!\rangle^{\Omega} = \langle\!\langle \eta \rangle\!\rangle^{\Omega}$.

Proof. Let $M \in \Lambda^o$, by Lemma 4 there exists $k \in \mathbb{N}$ such that $M\Omega^{\sim k} =_{\mathcal{B}} \Omega$. So we have:

$$\begin{array}{lll} \langle\!\langle \mathtt{I} \rangle\!\rangle^{\Omega} M & =_{\mathcal{B}} & \lambda x.[Mx, M\Omega x, \dots, M\Omega^{\sim k-1}x, \Omega, \dots] \\ & =_{\mathcal{B}} & \lambda x.[M(\mathtt{I}x), M\Omega(\mathtt{I}x), \dots, M\Omega^{\sim k-1}(\mathtt{I}x), \Omega, \dots] \\ & =_{\beta\eta} & \lambda x.[M(\eta_0 x), M\Omega(\eta_1 x), \dots, M\Omega^{\sim k-1}(\eta_{k-1}x), \Omega, \dots] =_{\mathcal{B}} \langle\!\langle \eta \rangle\!\rangle^{\Omega} M, \end{array}$$

where the third equality follows from $\mathtt{I} =_{\beta\eta} \eta_i$ for all $i \in \mathbb{N}$. Since M is an arbitrary closed λ-term, we can apply the ω-rule and conclude $\langle\!\langle \mathtt{I} \rangle\!\rangle^{\Omega} =_{\mathcal{B}\omega} \langle\!\langle \eta \rangle\!\rangle^{\Omega}$. ◀

As the variable y occurs in head-position in the terms of the streams $\langle\!\langle \eta \rangle\!\rangle^{\Omega} yx$ (resp. $\langle\!\langle \mathtt{I} \rangle\!\rangle^{\Omega} yx$), we can substitute for it a suitably modified projection that erases the Ω's and returns the n-th occurrence of x in $\langle\!\langle \mathtt{I} \rangle\!\rangle^{\Omega}$ (resp. $\eta_n(x)$ in $\langle\!\langle \eta \rangle\!\rangle^{\Omega}$).

▶ **Lemma 41.** *There is a closed λ-term* Eq *such that* $\mathsf{Eq}\,\mathtt{c}_n \langle\!\langle \mathtt{I} \rangle\!\rangle^{\Omega} =_{\mathcal{B}} \mathtt{I}$ *and* $\mathsf{Eq}\,\mathtt{c}_n \langle\!\langle \eta \rangle\!\rangle^{\Omega} =_{\mathcal{B}} \eta_n$.

Proof. Let Eq be a λ-term satisfying the recursive equation

$$\mathsf{Eq}\, n\, s = \mathsf{ifz}(n, \lambda z.s\mathtt{I}z\mathtt{K}, \mathsf{Eq}\,(\mathsf{pred}\, n)\,(\lambda zw.s(\mathtt{K}z)w\mathtt{F})).$$

By induction on n, we show $\mathsf{Eq}\,\mathtt{c}_n(\lambda yx.[y\Omega^{\sim i}(\eta_{i+k}x)]_{i\in\mathbb{N}}) =_{\mathcal{B}} \eta_{n+k}$ for all $n, k \in \mathbb{N}$. Note that $\eta_i \in \mathcal{I}^{\eta}$ entails $\lambda z.\eta_i z =_{\beta} \eta_i$. If $n = 0$ then $\mathsf{Eq}\,\mathtt{c}_0(\lambda yx.[y\Omega^{\sim i}(\eta_{i+k}x)]_{i\in\mathbb{N}})$ β-reduces to

$$\begin{array}{lllll} & \lambda z.(\lambda yx.[y\Omega^{\sim i}(\eta_{i+k}x)]_{i\in\mathbb{N}})\mathtt{I}z\mathtt{K} & =_{\beta} & \lambda z.[\Omega^{\sim i}(\eta_{i+k}z)]_{i\in\mathbb{N}}\mathtt{K} & \\ =_{\beta} & \lambda z.\mathtt{K}(\eta_k z)[\Omega^{\sim i+1}(\eta_{i+k+1}z)]_{i\in\mathbb{N}} & =_{\beta} & \lambda z.\eta_k z & =_{\beta} \eta_k. \end{array}$$

If $n > 0$ then we have $\mathsf{Eq}\,\mathtt{c}_n(\lambda yx.[y\Omega^{\sim i}(\eta_{i+k}x)]_{i\in\mathbb{N}}) =_{\beta} \mathsf{Eq}\,\mathtt{c}_{n-1}(\lambda yx.[y\Omega^{\sim i}(\eta_{i+k+1}x)]_{i\in\mathbb{N}})$ $=_{\text{(by Ind. Hyp.)}} \eta_{n-1+k+1} = \eta_{n+k}$. Indeed, easy calculations give:

$$\begin{array}{lllll} & \lambda zw.(\lambda yx.[y\Omega^{\sim i}(\eta_{i+k}x)]_{i\in\mathbb{N}})(\mathtt{K}z)w\mathtt{F} & =_{\beta} & \lambda zw.[\mathtt{K}z(\eta_k w), [\mathtt{K}z\Omega^{\sim i+1}(\eta_{i+k+1}w)]_{i\in\mathbb{N}}]\mathtt{F} \\ =_{\mathcal{B}} & \lambda zw.[z, [z\Omega^{\sim i}(\eta_{i+k+1}w)]_{i\in\mathbb{N}}]\mathtt{F} & =_{\beta} & \lambda zw.\mathtt{F}z[z\Omega^{\sim i}(\eta_{i+k+1}w)]_{i\in\mathbb{N}} \\ =_{\beta} & \lambda zw.[z\Omega^{\sim i}(\eta_{i+k+1}w)]_{i\in\mathbb{N}} & =_{\alpha} & \lambda yx.[y\Omega^{\sim i}(\eta_{i+k+1}x)]_{i\in\mathbb{N}}. \end{array}$$

Analogous calculations show $\mathsf{Eq}\,\mathtt{c}_n \langle\!\langle \mathtt{I} \rangle\!\rangle^{\Omega} =_{\mathcal{B}} \mathtt{I}$. ◀

▶ **Corollary 42.** $\mathcal{B}\omega \vdash \langle\!\langle \mathtt{I} \rangle\!\rangle = \langle\!\langle \eta \rangle\!\rangle$.

Proof. By Lemmas 41, 7 and 40: $\langle\!\langle \mathtt{I} \rangle\!\rangle =_{\mathcal{B}} [\mathsf{Eq}\,\mathtt{c}_n \langle\!\langle \mathtt{I} \rangle\!\rangle^{\Omega}]_{n\in\mathbb{N}} =_{\mathcal{B}\omega} [\mathsf{Eq}\,\mathtt{c}_n \langle\!\langle \eta \rangle\!\rangle^{\Omega}]_{n\in\mathbb{N}} =_{\mathcal{B}} \langle\!\langle \eta \rangle\!\rangle$. ◀

In Section 5.2 we have seen that, when $M \leq^{\eta} N$ holds, the λ-term $\Xi_{\iota}\lceil M \rceil \lceil N \rceil$ computes the Böhm tree of N from $\langle\!\langle \eta \rangle\!\rangle$ (Lemma 37) and the Böhm tree of M from $\langle\!\langle \mathtt{I} \rangle\!\rangle$ (Lemma 38), but now we have proved that $\langle\!\langle \eta \rangle\!\rangle =_{\mathcal{B}\omega} \langle\!\langle \mathtt{I} \rangle\!\rangle$. As a consequence, M and N are equal in $\mathcal{B}\omega$.

▶ **Theorem 43.** $\mathcal{B}\omega = \mathcal{H}^{+}$.

Proof. ($\subseteq$) By Lemma 22(ii), $\mathcal{B} \subseteq \mathcal{H}^+$ entails $\mathcal{B}\omega \subseteq \mathcal{H}^+\omega$. By Theorem 24 we have $\mathcal{H}^+\omega = \mathcal{H}^+$, so $\mathcal{B}\omega \subseteq \mathcal{H}^+$.

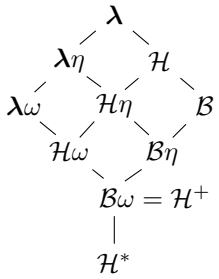
($\supseteq$) By Remark 10, it is enough to consider $M, N \in \Lambda^o$. If $\mathcal{H}^+ \vdash M = N$, then by Proposition 39 we have $M \leq^\eta P \geq^\eta N$ for $P = \Xi_\iota\lceil M\rceil\lceil N\rceil\langle\langle\eta\rangle\rangle$. Then we have:

$$\begin{array}{rll} M & =_{\mathcal{B}} & \Xi_\iota\lceil M\rceil\lceil P\rceil\langle\langle \mathtt{I}\rangle\rangle \quad \text{by Lemma 38} \\ & =_{\mathcal{B}\omega} & \Xi_\iota\lceil M\rceil\lceil P\rceil\langle\langle\eta\rangle\rangle \quad \text{by Corollary 42} \\ & =_{\mathcal{B}} & P \quad \text{by Lemma 37} \\ & =_{\mathcal{B}} & \Xi_\iota\lceil N\rceil\lceil P\rceil\langle\langle\eta\rangle\rangle \quad \text{by Lemma 37} \\ & =_{\mathcal{B}\omega} & \Xi_\iota\lceil N\rceil\lceil P\rceil\langle\langle \mathtt{I}\rangle\rangle \quad \text{by Corollary 42} \\ & =_{\mathcal{B}} & N \quad \text{by Lemma 38} \end{array}$$

We conclude that $\mathcal{B}\omega \vdash M = N$. ◀

This theorem disproves Sallé's conjecture (page 10) and settles one of the few open problems left in Barendregt's book. The next theorem should substitute Theorem 17.4.16 in [2].

▶ **Theorem 44.** *The following diagram indicates all inclusion relations of the λ-theories involved (if $\mathcal{T}_1$ is above $\mathcal{T}_2$, then $\mathcal{T}_1 \subsetneq \mathcal{T}_2$):*



Acknowledgments. The authors would like to thank Barendregt, Breuvart, Dezani-Ciancaglini, Lévy, Ronchi della Rocca and Ruoppolo for interesting discussions on $\mathcal{H}^+$.

References

1. Hendrik P. Barendregt. *Some extensional term models for combinatory logics and λ-calculi.* Ph.D. thesis, Utrecht Universiteit, the Netherlands, 1971.
2. Hendrik P. Barendregt. *The lambda calculus, its syntax and semantics.* Number 103 in Studies in Logic and the Foundations of Mathematics. North-Holland, second edition, 1984.
3. Hendrik P. Barendregt, Jan A. Bergstra, Jan Willem Klop, and Henri Volken. Degrees of sensible lambda theories. *Journal of Symbolic Logic*, 43(1):45–55, 1978.
4. Flavien Breuvart. On the characterization of models of $\mathcal{H}^*$. In Tom Henzinger and Dale Miller, editors, *CSL-LICS'14*, pages 24:1–24:10. ACM, 2014.
5. Flavien Breuvart, Giulio Manzonetto, Andrew Polonsky, and Domenico Ruoppolo. New results on Morris's observational theory. In Delia Kesner and Brigitte Pientka, editors, *Formal Structures for Computation and Deduction*, volume 52 of *LIPIcs*, pages 15:1–15:18. Schloss Dagstuhl, 2016.
6. Matt Brown and Jens Palsberg. Breaking through the normalization barrier: a self-interpreter for F-omega. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016*, pages 5–17. ACM, 2016.

7 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Patrick Sallé. Functional characterization of some semantic equalities inside lambda-calculus. In Hermann A. Maurer, editor, *Automata, Languages and Programming*, volume 71 of *Lecture Notes in Computer Science*, pages 133–146. Springer, 1979.

8 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Maddalena Zacchi. Type theories, normal forms and $\mathcal{D}_\infty$-lambda-models. *Information and Computation*, 72(2):85–116, 1987.

9 Pietro Di Gianantonio, Gianluca Franco, and Furio Honsell. Game semantics for untyped $\lambda\beta\eta$-calculus. In *Typed Lambda Calculi and Applications*, volume 1581 of *Lecture Notes in Computer Science*, pages 114–128. Springer, 1999.

10 Thomas Given-Wilson and Barry Jay. A combinatory account of internal structure. *J. Symb. Log.*, 76(3):807–826, 2011.

11 Xavier Gouy. *Étude des théories équationnelles et des propriétés algébriques des modèles stables du λ-calcul*. Thèse de doctorat, Université de Paris 7, 1995.

12 Martin Hyland. A survey of some useful partial order relations on terms of the λ-calculus. In *Lambda-Calculus and Computer Science Theory*, volume 37 of *Lecture Notes in Computer Science*, pages 83–95. Springer, 1975.

13 Martin Hyland. A syntactic characterization of the equality in some models for the λ-calculus. *Journal London Mathematical Society (2)*, 12(3):361–370, 1975/76.

14 Benedetto Intrigila and Monica Nesi. On structural properties of η-expansions of identity. *Inf. Proc. Lett.*, 87(6):327–333, 2003.

15 Benedetto Intrigila and Richard Statman. The omega rule is Π^0_2-hard in the $\lambda\beta$-calculus. In *Symposium on Logic in Computer Science (LICS 2004)*, pages 202–210. IEEE Computer Society, 2004.

16 Benedetto Intrigila and Richard Statman. The omega rule is Π^1_1-complete in the $\lambda\beta$-calculus. *Logical Methods in Computer Science*, 5(2), 2009.

17 Jean-Jacques Lévy. Approximations et arbres de Böhm dans le lambda-calcul. In Bernard Robinet, editor, *Lambda Calcul et Sémantique formelle des langages de programmation, Actes de la 6ème École de printemps d'Informatique théorique, La Châtre*, LITP-ENSTA, pages 239–257, 1978. (In French).

18 Stefania Lusin and Antonino Salibra. The lattice of λ-theories. *Journal of Logic and Computation*, 14(3):373–394, 2004.

19 Giulio Manzonetto. A general class of models of $\mathcal{H}^*$. In Rastislav Královic and Damian Niwinski, editors, *Mathematical Foundations of Computer Science 2009*, volume 5734 of *Lecture Notes in Computer Science*, pages 574–586. Springer, 2009.

20 Giulio Manzonetto and Domenico Ruoppolo. Relational graph models, Taylor expansion and extensionality. *Electronic Notes in Theoretical Computer Science*, 308:245–272, 2014.

21 Torben Æ. Mogensen. Efficient self-interpretations in lambda calculus. *J. Funct. Program.*, 2(3):345–363, 1992.

22 James H. Morris. *Lambda calculus models of programming languages*. PhD thesis, Massachusetts Institute of Technology, 1968.

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

24 Andrew Polonsky. Axiomatizing the Quote. In Marc Bezem, editor, *Computer Science Logic (CSL'11) – 25th International Workshop/20th Annual Conference of the EACSL*, volume 12 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 458–469, Dagstuhl, Germany, 2011. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

25 Simona Ronchi Della Rocca and Luca Paolini. *The Parametric λ-Calculus: a Metamodel for Computation*. EATCS Series. Springer, Berlin, 2004.

26 Patrick Sallé. Une extension de la théorie des types en λ-calcul. In Giorgio Ausiello and Corrado Böhm, editors, *Automata, Languages and Programming: Fifth Colloquium, Udine, Italy, July 17–21, 1978*, pages 398–410. Springer Berlin Heidelberg, 1978.

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

28 Paula Severi and Fer-Jan de Vries. An extensional Böhm model. In Sophie Tison, editor, *Rewriting Techniques and Applications: 13th International Conference, RTA 2002*, pages 159–173. Springer Berlin Heidelberg, 2002.

29 Paula Severi and Fer-Jan de Vries. The infinitary lambda calculus of the infinite η-Böhm trees. *Mathematical Structures in Computer Science*, 27(5):681–733, 2017.

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

Relating System F and $\lambda 2$: A Case Study in Coq, Abella and Beluga

Jonas Kaiser[1], Brigitte Pientka[2], and Gert Smolka[3]

1 **Saarland University, Saarbrücken, Germany**
`jkaiser@ps.uni-saarland.de`
2 **School of Computer Science, Montreal, QC, Canada**
`bpientka@cs.mcgill.ca`
3 **Saarland University, Saarbrücken, Germany**
`smolka@ps.uni-saarland.de`

Abstract

We give three formalisations of a proof of the equivalence of the usual, two-sorted presentation of System F and its single-sorted pure type system (PTS) variant $\lambda 2$. This is established by reducing the typability problem of F to $\lambda 2$ and vice versa. A key challenge is the treatment of variable binding and contextual information. The formalisations all share the same high level proof structure using relations to connect the type systems. They do, however, differ significantly in their representation and manipulation of variables and contextual information. In Coq, we use pure de Bruijn indices and parallel substitutions. In Abella, we use higher-order abstract syntax (HOAS) and nominal constants of the ambient reasoning logic. In Beluga, we also use HOAS but within contextual modal type theory. Our contribution is twofold. First, we present and compare a collection of machine-checked solutions to a non-trivial theoretical result. Second, we propose our proof as a benchmark, complementing the POPLmark and ORBI challenges by testing how well a given proof assistant or framework handles complex contextual information involving multiple type systems.

1998 ACM Subject Classification F.4.1 [Mathematical Logic] Lambda Calculus and Related Systems

Keywords and phrases Pure Type Systems, System F, de Bruijn Syntax, Higher-Order Abstract Syntax, Contextual Reasoning

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.21

1 Introduction

There are different presentations of "System F" in the literature and they are effectively considered equivalent, which is used to justify the transport of theoretical results between said presentations. The assumed notion of equivalence is primarily a reduction of the typability problem from one system to the other. While the existence of a suitable correspondence between the systems may appear likely or obvious, it turns out that actually proving it formally is surprisingly intricate. As long as the systems in question use the same expression syntax, the proofs are usually tedious but straightforward. If, on the other hand, not only the type systems, but also the syntactic languages differ, then establishing the correct correspondence becomes much more involved. The goal of this paper is to showcase various formalisation techniques to deal with the intricacies that arise in such an equivalence proof.

System F in its original form is due to Girard [14, 15], who introduced it in the context of proof theory. It was also independently discovered by Reynolds [26] as the polymorphic

$$\boxed{\mathsf{Ty_F}}\quad A, B ::= X \mid A \to B \mid \forall X.\, A \qquad \boxed{\mathsf{Tm_F}}\quad s, t ::= x \mid s\, t \mid \lambda x{:}A.\, s \mid s\, A \mid \Lambda X.\, s$$

$$\boxed{\mathsf{C}_\mathsf{F}^\mathsf{ty}}\quad \Delta ::= \emptyset \mid \Delta, X \qquad \boxed{\mathsf{C}_\mathsf{F}^\mathsf{tm}}\quad \Gamma ::= \bullet \mid \Gamma, x{:}A$$

$$\frac{X \in \Delta}{\Delta \Vdash_\mathsf{F}^\mathsf{ty} X} \qquad \frac{\Delta \Vdash_\mathsf{F}^\mathsf{ty} A \qquad \Delta \Vdash_\mathsf{F}^\mathsf{ty} B}{\Delta \Vdash_\mathsf{F}^\mathsf{ty} A \to B} \qquad \frac{\Delta, X \Vdash_\mathsf{F}^\mathsf{ty} A}{\Delta \Vdash_\mathsf{F}^\mathsf{ty} \forall X.\, A}\ X \notin \Delta \qquad \frac{\Gamma(x) = A \qquad \Delta \Vdash_\mathsf{F}^\mathsf{ty} A}{\Delta; \Gamma \Vdash_\mathsf{F}^\mathsf{tm} x : A}$$

$$\frac{\Delta; \Gamma \Vdash_\mathsf{F}^\mathsf{tm} s : A \to B \qquad \Delta; \Gamma \Vdash_\mathsf{F}^\mathsf{tm} t : A}{\Delta; \Gamma \Vdash_\mathsf{F}^\mathsf{tm} s\, t : B} \qquad \frac{\Delta; \Gamma, x{:}A \Vdash_\mathsf{F}^\mathsf{tm} s : B \qquad \Delta \Vdash_\mathsf{F}^\mathsf{ty} A}{\Delta; \Gamma \Vdash_\mathsf{F}^\mathsf{tm} \lambda x{:}A.\, s : A \to B}\ x \notin \operatorname{dom} \Gamma$$

$$\frac{\Delta; \Gamma \Vdash_\mathsf{F}^\mathsf{tm} s : \forall X.\, B \qquad \Delta \Vdash_\mathsf{F}^\mathsf{ty} A}{\Delta; \Gamma \Vdash_\mathsf{F}^\mathsf{tm} s\, A : B[A/X]} \qquad \frac{\Delta, X; \Gamma \Vdash_\mathsf{F}^\mathsf{tm} s : A}{\Delta; \Gamma \Vdash_\mathsf{F}^\mathsf{tm} \Lambda X.\, s : \forall X.\, A}\ X \notin \Delta$$

Figure 1 Two-sorted System F: types, terms, contexts, type formation and typing.

$$\boxed{\mathsf{Tm}_\lambda}\quad a, b, c, d, u ::= * \mid \Box \mid x \mid a\, b \mid \lambda x{:}a.\, b \mid \Pi x{:}a.\, b \qquad \boxed{\mathsf{C}_\lambda}\quad \Psi ::= \bullet \mid \Psi, x{:}a$$

$$\frac{}{\Psi \vdash_\lambda * : \Box} \qquad \frac{x{:}a \in \Psi \qquad \Psi \vdash_\lambda a : u}{\Psi \vdash_\lambda x : a} \qquad \frac{\Psi \vdash_\lambda a : u \qquad \Psi, x{:}a \vdash_\lambda b : *}{\Psi \vdash_\lambda \Pi x{:}a.\, b : *}\ x \notin \operatorname{dom} \Psi$$

$$\frac{\Psi \vdash_\lambda a : \Pi x{:}c.\, d \qquad \Psi \vdash_\lambda b : c}{\Psi \vdash_\lambda a\, b : d[b/x]} \qquad \frac{\Psi \vdash_\lambda a : u \qquad \Psi, x{:}a \vdash_\lambda b : c \qquad \Psi, x{:}a \vdash_\lambda c : *}{\Psi \vdash_\lambda \lambda x{:}a.\, b : \Pi x{:}a.\, c}\ x \notin \operatorname{dom} \Psi$$

Figure 2 PTS: terms, contexts, and the type system λ2; u ranges over the universes $*$ and $\Box$.

λ-calculus. For the purpose of this paper we consider two presentations that differ sufficiently to demonstrate the various complications. The first, called F and shown in Figure 1, is the common two-sorted presentation, as for example given by Harper [17]. The second, given in Figure 2, is the single-sorted pure type system (PTS) λ2, which appears as a corner in Barendregt's λ-cube [5].

In [13], Geuvers gives a proof sketch that valid typing judgements can be translated between these two presentations. In [19] we then gave the first proof, machine-checked in Coq, of the full reduction result. It relies on syntactic translation functions and the construction of an intermediate, F-like type system for the PTS syntax. The proof presented here is a lot simpler for a number of reasons. We use relations on the two syntactic languages rather than translation functions, so we do not have to concern ourselves with cancellation laws to establish the reductions. We can further establish the correspondence directly, which completely bypasses the need for the intermediate type system.

The relations that precisely establish the correspondence of our two systems are realised as inductive predicates (Figure 3). The main advantage over the functional approach is that it allows us to focus on the meaningful, well-typed fragments of the two systems. Despite these simplifications we still have to face the metaphorical elephant in the room, namely variable binding and context manipulation. More precisely, we have to represent and handle contexts that track various kinds of information, like the set of defined variables, their associated types, their correspondence to other variables, and so on. To make the situation fully explicit: our contexts are dependent sequences of dependent records.

In the following we present three formalisations. They all follow the same basic proof structure, outlined in Section 2, but they each deal with the complexities of contextual

$$\boxed{\mathsf{C}^{\mathsf{ty}}_{\mathsf{R}}} \quad \Theta ::= \bullet \mid \Theta, (X, y) \qquad \boxed{\mathsf{C}^{\mathsf{tm}}_{\mathsf{R}}} \quad \Sigma ::= \bullet \mid \Sigma, (x, y)$$

$$\frac{(X, y) \in \Theta}{\Theta \vdash X \sim y} \qquad \frac{\Theta \vdash A \sim a \qquad \Theta \vdash B \sim b}{\Theta \vdash A \to B \sim \Pi y{:}a.\, b}\ y \notin \Theta \qquad \frac{\Theta, (X, y) \vdash A \sim a}{\Theta \vdash \forall X.\, A \sim \Pi y{:}*.\, a}\ X, y \notin \Theta$$

$$\frac{(x, y) \in \Sigma}{\Theta; \Sigma \vdash x \approx y} \qquad \frac{\Theta; \Sigma \vdash s \approx a \qquad \Theta; \Sigma \vdash t \approx b}{\Theta; \Sigma \vdash s\, t \approx a\, b} \qquad \frac{\Theta; \Sigma \vdash s \approx a \qquad \Theta \vdash A \sim b}{\Theta; \Sigma \vdash s\, A \approx a\, b}$$

$$\frac{\Theta \vdash A \sim a \qquad \Theta; \Sigma, (x, y) \vdash s \approx b}{\Theta; \Sigma \vdash \lambda x{:}A.\, s \approx \lambda y{:}a.\, b}\ x, y \notin \Theta, \Sigma \qquad \frac{\Theta, (X, y); \Sigma \vdash s \approx a}{\Theta; \Sigma \vdash \Lambda X.\, s \approx \lambda y{:}*.\, a}\ X, y \notin \Theta, \Sigma$$

Figure 3 Inductive characterisation of $\sim$ and $\approx$; Θ and Σ track related type and term variables.

information and variable binding in different ways. In Sections 3, 4 and 5 we discuss in detail how this is managed, in order of seniority, in Coq [6], Abella [4] and respectively Beluga [25].

The Coq proof assistant, oldest among the three, is a general purpose theorem prover based on constructive type theory with no particular built-in support for meta-theoretical reasoning. Since all required structures have to be handled manually, library support is essential. In our case we use the Autosubst framework [28] that allows us to elegantly work with pure de Bruijn syntax and parallel substitutions.

In contrast, both Abella and Beluga allow us to work with higher-order abstract syntax (HOAS) [21], albeit in two rather different background logics. Both systems are designed explicitly with meta-theoretical reasoning in mind, with suitable support built into the foundation of each system. The core design of Abella is based around proof search and relational specifications. It predates the POPLmark challenge [2], which shaped the following decade of formalised meta theory. Meanwhile Beluga, youngest among the three, revisited and and extended the Twelf logical framework [22] to directly support first-class syntactic contexts and substitutions in its type-theoretic foundation. This lead to the notion of contextual objects. Our three developments demonstrate how the various system designs affect a given formalisation effort.

While the comparison of proofs from different systems in terms of code lines is only marginally meaningful, we were surprised to find that all three developments each take approximately 500 loc, with Beluga slightly on the shorter side and Coq somewhat on the longer. This, however, only covers establishing the correspondence itself. The systems require vastly different amounts of code to establish the separate meta theories for the two discussed type systems, due to different levels of background support.

Contributions of the paper

1. We present and compare three different machine checked formalisations of the technically intricate reduction of typability from System F to the PTS $\lambda 2$ and vice versa.[1]
2. We propose that our equivalence proof serves as a benchmark for reasoning about and relating multiple type systems and languages involving variable binding. The key aspect of this benchmark is the representation and manipulation of contexts that track multiple kinds of information and exhibit complex dependency structures. As such it can be seen as a complement to the benchmarks proposed in [10, 9] and the POPLmark challenge [2].

[1] The accompanying developments can be found at `https://www.ps.uni-saarland.de/extras/fscd17/`.

2 Equivalence

The core challenge of the proof is the fact that F clearly distinguishes types and terms with separate syntactic sorts, $\mathsf{Ty_F}$ and $\mathsf{Tm_F}$ respectively, while $\lambda 2$ merges these into a single syntactic sort Tm_λ. The distinction still exists in $\lambda 2$ but it is semantically imposed through the type system, rather than at the level of syntax. Further consequences are the existence of two variable scopes in F, with separate abstraction and application mechanisms, while the same concepts are uniformly represented in $\lambda 2$ for a single variable scope. This extends to the formation of function spaces as well. For an in-depth discussion of the mismatches between the two systems see [19]. In essence, the two systems differ in how explicit and readily available certain structural properties are. One half of the proof will thus have to reestablish implicit structures, which, as one would expect, is harder than removing it. This will lead to a certain asymmetry in proof effort for seemingly symmetrical lemma statements.

The basic idea of the proof presented here is to construct two relations $\sim$ and $\approx$ that put the types and respectively the terms of the two languages in correspondence (see Figure 3). To obtain the desired equivalence results, we have to demonstrate that these relations exhibit the following properties:

1. $\sim$ is functional and injective.
2. $\sim$ is left-total and type-formation preserving on the well-formed types of F.
3. $\sim$ is right-total and type-formation preserving on the propositions of $\lambda 2$. A proposition of $\lambda 2$ is any term a such that $\vdash_\lambda a : *$ holds.
4. $\approx$ is functional and injective.
5. $\approx$ is left-total and typing preserving on the well-typed terms of F.
6. $\approx$ is right-total and typing preserving on the proofs of $\lambda 2$. A proof of $\lambda 2$ is any term b such that $\vdash_\lambda b : a$ holds for a a proposition of $\lambda 2$.

We can now formulate, and easily prove, the following equivalences:

▶ **Theorem 1** (Reductions from F to $\lambda 2$)**.**

$$\vdash^{\mathsf{tv}}_{\mathsf{F}} A \iff \exists! a.\ \vdash A \sim a \ \wedge\ \vdash_\lambda a : *$$
$$\vdash^{\mathsf{tm}}_{\mathsf{F}} s : A \iff \exists! b a.\ \vdash s \approx b \ \wedge\ \vdash A \sim a \ \wedge\ \vdash_\lambda b : a \ \wedge\ \vdash_\lambda a : *$$

Proof. The forward directions are simply the corresponding left-to-right preservation and left-totality results of $\approx$ and $\sim$. Uniqueness follows from functionality. For the inverse direction we use preservation (here from right to left) and uniqueness. ◀

▶ **Theorem 2** (Reductions from $\lambda 2$ to F)**.**

$$\vdash_\lambda a : * \iff \exists! A.\ \vdash A \sim a \ \wedge\ \vdash^{\mathsf{tv}}_{\mathsf{F}} A$$
$$\vdash_\lambda b : a \ \wedge\ \vdash_\lambda a : * \iff \exists! s A.\ \vdash s \approx b \ \wedge\ \vdash A \sim a \ \wedge\ \vdash^{\mathsf{tm}}_{\mathsf{F}} s : A$$

Proof. Dual to the previous result. ◀

For the remainder of the paper, we focus on how F, $\lambda 2$ and the two relations $\sim$ and $\approx$ are represented in our three proof systems, and how the six main properties of the relations are obtained. Establishing Theorems 1 and 2 is in each case routine and hence not presented in detail.

$$A, B ::= x_{\mathsf{ty}} \mid A \to B \mid \forall.\, A \qquad s, t ::= x_{\mathsf{tm}} \mid s\, t \mid \lambda A.\, s \mid s\, A \mid \Lambda.\, s \qquad x, N : \mathbb{N}$$

$$\frac{x < N}{N \Vdash^{\mathsf{ty}}_{\mathsf{F}} x_{\mathsf{ty}}} \qquad \frac{N \Vdash^{\mathsf{ty}}_{\mathsf{F}} A \quad N \Vdash^{\mathsf{ty}}_{\mathsf{F}} B}{N \Vdash^{\mathsf{ty}}_{\mathsf{F}} A \to B} \qquad \frac{N + 1 \Vdash^{\mathsf{ty}}_{\mathsf{F}} A}{N \Vdash^{\mathsf{ty}}_{\mathsf{F}} \forall.\, A} \qquad \frac{\Gamma_x = A \quad N \Vdash^{\mathsf{ty}}_{\mathsf{F}} A}{N; \Gamma \Vdash^{\mathsf{tm}}_{\mathsf{F}} x_{\mathsf{tm}} : A} \qquad \frac{N; \Gamma \Vdash^{\mathsf{tm}}_{\mathsf{F}} s : \forall.\, A \quad N \Vdash^{\mathsf{ty}}_{\mathsf{F}} B}{N; \Gamma \Vdash^{\mathsf{tm}}_{\mathsf{F}} s\, B : A[B \cdot \mathsf{id}]}$$

$$\frac{N + 1; \Gamma[\uparrow] \Vdash^{\mathsf{tm}}_{\mathsf{F}} s : A}{N; \Gamma \Vdash^{\mathsf{tm}}_{\mathsf{F}} \Lambda.\, s : \forall.\, A} \qquad \frac{N; \Gamma, A \Vdash^{\mathsf{tm}}_{\mathsf{F}} s : B \quad N \Vdash^{\mathsf{ty}}_{\mathsf{F}} A}{N; \Gamma \Vdash^{\mathsf{tm}}_{\mathsf{F}} \lambda A.\, s : A \to B} \qquad \frac{N; \Gamma \Vdash^{\mathsf{tm}}_{\mathsf{F}} s : A \to B \quad N; \Gamma \Vdash^{\mathsf{tm}}_{\mathsf{F}} t : A}{N; \Gamma \Vdash^{\mathsf{tm}}_{\mathsf{F}} s\, t : B}$$

Figure 4 System F – de Bruijn encoding in Coq; term variable contexts Γ are lists of types.

$$a, b, c, d ::= * \mid \Box \mid x \mid a\, b \mid \lambda a.\, b \mid \Pi a.\, b \qquad x : \mathbb{N}$$

$$\frac{}{0 : a[\uparrow] \in_\lambda \Psi, a} \qquad \frac{x : a \in_\lambda \Psi}{(x+1) : a[\uparrow] \in_\lambda \Psi, b} \qquad \frac{}{\Psi \vdash_\lambda * : \Box} \qquad \frac{x : a \in_\lambda \Psi \quad \Psi \vdash_\lambda a : u}{\Psi \vdash_\lambda x : a}$$

$$\frac{\Psi \vdash_\lambda a : u \quad \Psi, a \vdash_\lambda b : *}{\Psi \vdash_\lambda \Pi a.\, b : *} \qquad \frac{\Psi \vdash_\lambda a : \Pi c.\, d \quad \Psi \vdash_\lambda b : c}{\Psi \vdash_\lambda a\, b : d[b \cdot \mathsf{id}]} \qquad \frac{\Psi \vdash_\lambda a : u \quad \Psi, a \vdash_\lambda b : c \quad \Psi, a \vdash_\lambda c : *}{\Psi \vdash_\lambda \lambda a.\, b : \Pi a.\, c}$$

Figure 5 $\lambda 2$ – de Bruijn encoding in Coq; dependent contexts Ψ are lists of terms.

3 Coq

For the Coq proof we reuse the pure de Bruijn encoding of the two systems and the corresponding meta theories developed in [19], with major support from the Autosubst framework [28]. The language definitions are given in Figures 4 and 5. The main feature of de Bruijn syntax is the absence of variable names. Variables are instead represented as numerical indices, where n references the nth enclosing binder of the correct scope. Dangling indices represent free variables that instead reference positions in an enclosing context, indexed from right to left. Note that we preserve the dot as a notational device to uniformly indicate the presence of a binding constructor, even if nothing remains to the left of it (e.g. $\Lambda.\, s$). It marks the precise spot where substitutions and indices have to be adjusted. The application of a parallel substitution σ to a term s is written $s[\sigma]$ where σ is a function from $\mathbb{N}$ that acts on all free variables of s at once. The Autosubst framework provides a normalisation procedure for such terms with applied substitutions. The existence of computable normal forms was demonstrated in [27]. Context morphism lemmas (CML) are a useful proof device to reason about judgements over pure de Bruijn syntax. In the following we will only focus on those aspects that have an immediate impact on our present proof. For an in-depth discussion of the interaction of pure de Bruijn syntax, parallel substitutions and CMLs we refer to [7, 27, 28, 1, 16, 19]. A brief overview is also given in Appendix A.

In the definition of F in Figure 4, observe how the type variable context Δ degenerates to a plain natural number N. It is taken as an exclusive upper bound to the admissible type variable indices, hence $N = 0$ represents the empty context. The term variable context Γ is simply a list of types, since free variables are coded as context positions. The substitution $B \cdot \mathsf{id}$ used in the type specialisation rule maps the free index 0 to B and lowers all other indices by 1.

We further observe that for $\lambda 2$, defined in Figure 5, context lookup is characterised inductively: $x : a \in_\lambda \Psi$. The need for this arises from the fact that the PTS contexts are

dependent as well as the more general issue that in a de Bruijn setting, terms are not stable under context modifications. Hence, upon extraction of a term a from context Γ, all free variables of a have to be adjusted by an amount that depends on the position of a in Γ. The given inductive characterisation elegantly handles this complication.

The first equivalence proof. Similar to the way typing contexts are explicitly represented as lists of terms or types, we are going to track explicitly which variables are related in our definition of our relations $\sim^R$ and $\approx^R_S$. The relational parameters R and S correspond to the contexts Θ and Σ from Figure 3:

$$\frac{x\,R\,y}{x_{\mathsf{ty}}\ \sim^R\ y} \qquad \frac{A\ \sim^R\ a \quad B\ \sim^{R^{\Uparrow}}\ b}{A \to B\ \sim^R\ \Pi a.\,b} \qquad \frac{A\ \sim^{R^{\mathsf{ext}}}\ a}{\forall.\,A\ \sim^R\ \Pi *.\,a} \qquad \frac{x\,S\,y}{x_{\mathsf{tm}}\ \approx^R_S\ y}$$

$$\frac{s\ \approx^R_S\ a \quad t\ \approx^R_S\ b}{s\,t\ \approx^R_S\ a\,b} \qquad \frac{s\ \approx^R_S\ a \quad A\ \sim^R\ b}{s\,A\ \approx^R_S\ a\,b} \qquad \frac{A\ \sim^R\ a \quad s\ \approx^{R^{\Uparrow}}_{S^{\mathsf{ext}}}\ b}{\lambda A.\,s\ \approx^R_S\ \lambda a.\,b} \qquad \frac{s\ \approx^{R^{\mathsf{ext}}}_{S^{\Uparrow}}\ a}{\Lambda.\,s\ \approx^R_S\ \lambda *.\,a}$$

The parameters R and S track pairs of indices of type, and respectively, term variables. We technically represent them as lists of type $\mathsf{list}\,(\mathsf{var} \times \mathsf{var})$ and use $x\,R\,y$ to denote that the pair (x, y) is in R. The interesting part of this definition is how these auxiliary parameters have to be modified when binders are traversed, which we denoted above by $R^{\Uparrow}$ and R^{ext}. In order to precisely define these operations, let us recall the required action on a parallel substitution σ that is pushed underneath a binder:

$$(\forall.\,A)[\sigma] \qquad \longmapsto \qquad \forall.\,A[0 \cdot \sigma \circ \uparrow]$$

The $[0 \cdot \_]$ part ensures that any index referencing the presently traversed binder is preserved as such. Meanwhile the $[\_ \cdot \sigma \circ \uparrow]$ part ensures that every index $n + 1$ is mapped to $\sigma(n)[\uparrow]$, where the shift $\uparrow$ ensures that no free variables in the range of σ are accidentally captured by the traversed binder.

In our correspondence proof we traverse binders *almost* in lockstep. For the simple cases where we have a binder on both sides of the relation, and moreover, the bound variables actually correspond according to the information tracked in R, we define, analogously to the binder traversal for substitutions:

$$R^{\mathsf{ext}} := (0, 0) :: \mathsf{bimap} \uparrow \uparrow R$$

where $\mathsf{bimap}\ f\ g\ R$ simply applies f to all left projections of R and g to all right projections.

The other possible scenario has a binder on the $\lambda 2$ side that has no counterpart in F with respect to the contextual information in R, like the *not-really dependent* PTS product that corresponds to an arrow type in F. As a consequence of this spurious binding, the $\lambda 2$ indices in R have to be shifted relative to their F counterparts. This one-sided index adjustment is obtained with

$$R^{\Uparrow} := \mathsf{bimap}\ \mathsf{id} \uparrow R$$

▶ **Fact 3.** *Both R^{ext} and $R^{\Uparrow}$ preserve injectivity and functionality of R.* ◀

▶ **Lemma 4.** *The type relation $\sim^R$ is injective/functional, whenever R is injective/functional.*

Proof. Straightforward inductions using Fact 3. ◀

To obtain the same result for $\approx_S^R$ we additionally rely on R and S having disjoint ranges, that is, no PTS variable is considered related to both a type and a term variable. We denote this by $R \| S$.

▶ **Fact 5.** *The property $R\|S$ is preserved under extending one relation and shifting the other, that is w.l.o.g.:* $R\|S \implies R^{\Uparrow}\|S^{\mathsf{ext}}$. ◀

▶ **Lemma 6.** *Disjointedness of ranges lifts from variable relations R and S to $\sim^R$ and $\approx_S^R$:*

$$R\|S \implies A \sim^R a \implies s \approx_S^R a \implies \bot$$

Proof. By induction on $A \sim^R a$ and discriminating on $s \approx_S^R a$, using Fact 5. ◀

▶ **Lemma 7.** *The term relation $\approx_S^R$ is functional, whenever R and S are functional. It is injective, whenever R and S are injective and $R\|S$ holds.*

Proof. Straightforward inductions. Injectivity relies on the premise $R\|S$ and Lemma 6 to discharge non-matching applications. Subderivations for $\sim^R$ are handled with Lemma 4. ◀

Proving the left and right totality and preservation results is slightly more interesting, as we have to generalise to open judgements and non-empty contexts. We achieve this with suitable proof invariants that are adapted from the notion of *generalised context morphisms* laid out in [19]. The key difference is that instead of a renaming ξ that maps from one context to another, we here consider a relation on variables that places two contexts in correspondence. All invariants are set up such that they vacuously hold when the initial context happens to be empty. We start with type formation and the direction from F to $\lambda 2$:

$$N \xrightarrow{R} \Psi := \forall x < N.\ \exists y.\ x\,R\,y \wedge y{:}* \in_\lambda \Psi$$

▶ **Fact 8.** *The invariant $N \xrightarrow{R} \Psi$ is preserved under corresponding extensions:*

$$N \xrightarrow{R} \Psi \implies N \xrightarrow{R^{\Uparrow}} \Psi, a \qquad\qquad N \xrightarrow{R} \Psi \implies N+1 \xrightarrow{R^{\mathsf{ext}}} \Psi, *$$ ◀

▶ **Lemma 9.** *The type-relation $\sim^R$ is left-total and preserves type formation:*

$$N \mid^{\mathsf{ty}}_{\mathsf{F}} A \implies \forall R\,\Psi.\ N \xrightarrow{R} \Psi \implies \exists a.\ A \sim^R a \wedge \Psi \vdash_\lambda a : *$$

Proof. By induction on $N \mid^{\mathsf{ty}}_{\mathsf{F}} A$. The two binder cases use Fact 8. ◀

For the inverse direction we establish preservation of type formation and right totality along the following invariant:

$$N \xleftarrow{R} \Psi := \forall y.\ y{:}* \in_\lambda \Psi \implies \exists x.\ x\,R\,y \wedge x < N$$

▶ **Fact 10.** *The invariant $N \xleftarrow{R} \Psi$ is preserved under corresponding extensions:*

$$N \xleftarrow{R} \Psi \implies \Psi \vdash_\lambda a : * \implies N \xleftarrow{R^{\Uparrow}} \Psi, a \qquad N \xleftarrow{R} \Psi \implies N+1 \xleftarrow{R^{\mathsf{ext}}} \Psi, *$$ ◀

▶ **Lemma 11.** *The type-relation $\sim^R$ is right-total and preserves type formation:*

$$\Gamma \vdash_\lambda a : * \implies \forall R\,N.\ N \xleftarrow{R} \Gamma \implies \exists A.\ A \sim^R a \wedge N \mid^{\mathsf{ty}}_{\mathsf{F}} A$$

Proof. Induction on $\Gamma \vdash_\lambda a : *$, using Fact 10. One complication is the disambiguation of a given PTS-product $\Pi a.\, b$, where a is known to live in some universe u. Discriminating on u allows us to correctly choose either an arrow type $A \to B$, or a universal quantification $\forall.\, B$. Further requirements are the degeneracy of the universe $\Box$ ($*$ is the only inhabitant of $\Box$), as well as propagation and substitutivity for $\vdash_\lambda$. ◀

The preservation and totality results for $\approx_S^R$ make the overhead for explicitly tracking contextual information most apparent. Since some of the typing rules for applications ascribe types that are constructed from a non-trivial substitution operation, we require substitutivity results for the judgements under consideration; in particular, β-substitutivity for $\sim^R$:

▶ **Fact 12.** *The type relation $\sim^R$ is closed under β-substitutions:*

$$B \sim^R b \implies A \sim^{R^{\mathsf{ext}}} a \implies A[B \cdot \mathsf{id}] \sim^R a[b \cdot \mathsf{id}]$$ ◀

The proof of this fact is a lengthy but straightforward construction that first generalises the two concrete β-substitutions to arbitrary parallel substitutions σ and τ. The result is still not provable directly, as closure of $\sim^R$ under weakening is needed. This in turn is generalised to a provable statement for arbitrary renamings ξ and ζ in place of σ and τ. In essence, we establish a CML for $\sim^R$.

We can now tackle the technically most intricate part of the proof. The invariant for preservation of typing from F to $\lambda 2$ is

$$\Gamma \xrightarrow[S]{R} \Psi := \forall xA.\, \Gamma_x = A \implies \exists ya.\, A \sim^R a \wedge x\, S\, y \wedge y{:}a \in_\lambda \Psi$$

▶ **Fact 13.** *The invariant $\Gamma \xrightarrow[S]{R} \Psi$ is preserved under corresponding extensions:*

$$\Gamma \xrightarrow[S]{R} \Psi \implies A \sim^R a \implies \Gamma, A \xrightarrow[S^{\mathsf{ext}}]{R^{\Uparrow}} \Psi, a \qquad \Gamma \xrightarrow[S]{R} \Psi \implies \Gamma[\uparrow] \xrightarrow[S^{\Uparrow}]{R^{\mathsf{ext}}} \Psi, *$$ ◀

▶ **Lemma 14.** *The term relation $\approx_S^R$ is left-total and preserves typing.*

$$N; \Gamma \vdash_{\mathsf{F}}^{\mathsf{tm}} s : A \implies \forall RS\Psi.\, R\ \mathsf{func} \implies N \xrightarrow{R} \Psi \implies \Gamma \xrightarrow[S]{R} \Psi \implies$$
$$\exists ba.\, A \sim^R a \wedge s \approx_S^R b \wedge \Psi \vdash_\lambda b : a \wedge \Psi \vdash_\lambda a : *$$

Proof. By induction on $N; \Gamma \vdash_{\mathsf{F}}^{\mathsf{tm}} s : A$. Both the invariant $\Gamma \xrightarrow[S]{R} \Psi$, as well as Lemma 9, are used to obtain related types in the variable case. Functionality of R allows us to equate these. ◀

The final part is the preservation of typing from $\lambda 2$ to F. Here we use:

$$\Gamma \xleftarrow[S]{R} \Psi := \forall ya.\, y{:}a \in_\lambda \Psi \implies \Psi \vdash_\lambda a : * \implies \exists xA.\, A \sim^R a \wedge x\, S\, y \wedge x{:}A \in \Gamma$$

▶ **Fact 15.** *The invariant $\Gamma \xleftarrow[S]{R} \Psi$ is preserved under corresponding extensions:*

$$\Gamma \xleftarrow[S]{R} \Psi \implies A \sim^R a \implies \Gamma, A \xleftarrow[S^{\mathsf{ext}}]{R^{\Uparrow}} \Psi, a \qquad \Gamma \xleftarrow[S]{R} \Psi \implies \Gamma[\uparrow] \xleftarrow[S^{\Uparrow}]{R^{\mathsf{ext}}} \Psi, *$$ ◀

▶ **Lemma 16.** *The term relation $\approx_S^R$ is right-total and preserves typing:*

$$\Psi \vdash_\lambda a : * \implies \Psi \vdash_\lambda b : a \implies \forall RSN\Gamma.\, R\ \mathsf{inj} \implies N \xleftarrow{R} \Psi \implies \Gamma \xleftarrow[S]{R} \Psi \implies$$
$$\exists sA.\, A \sim^R a \wedge s \approx_S^R b \wedge N; \Gamma \vdash_{\mathsf{F}}^{\mathsf{tm}} s : A \wedge N \vdash_{\mathsf{F}}^{\mathsf{ty}} A$$

$$\frac{A\ \mathsf{ty} \qquad B\ \mathsf{ty}}{(A \to B)\ \mathsf{ty}} \qquad \frac{\mathbf{\Pi} x.\ x\ \mathsf{ty} \Rrightarrow A\langle x\rangle\ \mathsf{ty}}{(\forall.\ A)\ \mathsf{ty}} \qquad \frac{s :_{\mathsf{F}} \forall.\ B \qquad A\ \mathsf{ty}}{s\ A :_{\mathsf{F}} B\langle A\rangle} \qquad \frac{s :_{\mathsf{F}} A \to B \qquad t :_{\mathsf{F}} A}{s\ t :_{\mathsf{F}} B}$$

$$\frac{\mathbf{\Pi} x.\ x\ \mathsf{ty} \Rrightarrow s\langle x\rangle :_{\mathsf{F}} A\langle x\rangle}{\Lambda.\ s :_{\mathsf{F}} \forall.\ A} \qquad \frac{A\ \mathsf{ty} \qquad \mathbf{\Pi} x.\ x :_{\mathsf{F}} A \Rrightarrow s\langle x\rangle :_{\mathsf{F}} B}{\lambda A.\ s :_{\mathsf{F}} A \to B}$$

Figure 6 HOAS specification of F in Abella.

$$\frac{}{\mathcal{U}\ \Box} \qquad \frac{}{\mathcal{U}\ *} \qquad \frac{}{* :_\lambda \Box} \qquad \frac{a :_\lambda \Pi c.\ d \qquad b :_\lambda c}{a\ b :_\lambda d\langle b\rangle} \qquad \frac{a :_\lambda u \qquad \mathcal{U}\ u \qquad \mathbf{\Pi} x.\ x :_\lambda a \Rrightarrow b\langle x\rangle :_\lambda *}{\Pi a.\ b :_\lambda *}$$

$$\frac{a :_\lambda u \qquad \mathcal{U}\ u \qquad \mathbf{\Pi} x.\ x :_\lambda a \Rrightarrow c\langle x\rangle :_\lambda * \qquad \mathbf{\Pi} x.\ x :_\lambda a \Rrightarrow b\langle x\rangle :_\lambda c\langle x\rangle}{\lambda a.\ b :_\lambda \Pi a.\ c}$$

Figure 7 HOAS specification of $\lambda 2$ in Abella.

Proof. By induction on $\Psi \vdash_\lambda b : a$. The cases are mostly analogue to the previous result. Injectivity of R is required for the variable case. Note that we need to discriminate on the universes of product domains again (cf. Lemma 11), here to disambiguate the unified abstractions and applications correctly. ◀

At this point we can make an interesting observation. The above proof demonstrates that the CML proof pattern not only generalises to a multi system setting [19] but also to relations in place of functional correspondences. This is what allowed us to quickly generate all the contextual invariants needed for the various results.

4 Abella

Abella supports the use of higher-order abstract syntax (HOAS) [21]. The main idea is to delegate variable binding at the object level to binding at the meta level. Take for example the term constructor $\mathsf{lam} : (\mathsf{tm} \to \mathsf{tm}) \to \mathsf{tm}$ that yields an abstraction of the untyped λ-calculus. The s in $\mathsf{lam}\ s$ is a function of the meta level. Substitution at the object level is implemented as application at the meta level, which we denote by $s\langle t\rangle$ to distinguish it from the various object level applications.

Reasoning about HOAS encodings inductively is somewhat complicated, as terms do not remain closed and escape into the host theory. To get around this, Abella is designed around the so-called two-level logic approach. The lower *specification level*, essentially verbatim λProlog, is used to encode the object languages and their associated judgements. One then reasons about these encodings at the *meta level*, using the logic $\mathcal{G}$, which is the intuitionistic predicative fragment of Church's simple type theory, extended with natural induction, (co)inductive predicates and nominal quantification ($\nabla x.\ s$) [11, 12]. The axioms of $\mathcal{G}$ ensure that a ∇-quantified identifier is fresh for everything bound above it. The faithful representation of object level variables relies on this freshness guarantee.

The two levels are connected with a special inductive predicate that embeds the derivation of a λProlog judgement J from hypotheses $L = I_0, \ldots, I_n$ into $\mathcal{G}$, written $\{L \vdash J\}$, or simply $\{J\}$ in the absence of assumptions. Note that λProlog supports hypothetical ($J_1 \Rrightarrow J_2$)

and locally quantified ($\mathbf{\Pi} x.\ J[x]$) premises, which the embedding treats as follows:

$$\{L \vdash J_1 \Rightarrow\!\blacktriangleright\ J_2\} \quad \rightsquigarrow \quad \{L, J_1 \vdash J_2\} \qquad \{L \vdash \mathbf{\Pi} x.\ J[x]\} \quad \rightsquigarrow \quad \nabla x.\{L \vdash J[x]\}$$

For the quantification case, observe that the context L is usually bound at the outermost level, hence x is guaranteed to be fresh for L, satisfying the usual side condition for context extension rules. This process is often referred to as *mobility of binders*: object level binders are represented using λProlog quantification, which in turn is mapped to ∇-quantification in $\mathcal{G}$ and eventually opened with nominal constants n_i.

▶ **Fact 17.** *The embedding $\{L \vdash J\}$ satisfies cut and a nominal instantiation principle:*

$$\{L \vdash I\} \implies \{L, I \vdash J\} \implies \{L \vdash J\} \tag{cut}$$
$$\forall s : \mathsf{typeof}\ n_i.\ \{L[n_i] \vdash J[n_i]\} \implies \{L[s] \vdash J[s]\} \tag{inst}$$

Both are exposed as proof tactics to the user. ◀

Note that Fact 17 provides substitutivity for the various judgements of our object languages. It is also worth noting that only the judgements, but not the specification level types like Tm_λ and Ty_F, are embedded, so we cannot induct directly on our syntax definitions.

At this point it should be straightforward to understand the HOAS representation of our two systems given in Figures 6 and 7.[2] For F, the judgements $A\,\mathsf{ty}$ and $s :_\mathsf{F} A$ encode type formation and respectively typing. For λ2, the judgement $\mathcal{U}\,u$ is used to recognise universes, while $a :_\lambda b$ represents PTS typing.

As soon as we try to prove structural results about these definitions we are faced with a problem. Consider the following inversion principle for arrow type formation in F:

$$\{L \vdash A \to B\,\mathsf{ty}\} \implies \{L \vdash A\,\mathsf{ty}\} \wedge \{L \vdash B\,\mathsf{ty}\}$$

The premise may hold not only due to structural reasons, as claimed by the lemma, but also due to backchaining and $A \to B\,\mathsf{ty} \in L$. The problem is that L is too general and may contain arbitrary judgements, not even necessarily related to type formation. Hence we need to somehow constrain L to only contain judgements of the form $n_i\,\mathsf{ty}$, where the n_i are nominals representing variables. Similarly we want to constrain typing contexts to only contain judgements of the form $n_i\,\mathsf{ty}$ or $n_i :_\mathsf{F} A$. We specify this notion of well-formed contexts with auxiliary inductive $\mathcal{G}$-predicates. The well-formedness predicate for F typing contexts, written $\mathbb{C}^{\mathsf{tm}}_\mathsf{F}(-)$, is defined as:

$$\frac{}{\mathbb{C}^{\mathsf{tm}}_\mathsf{F}(\bullet)} \qquad \frac{\mathbb{C}^{\mathsf{tm}}_\mathsf{F}(L)}{\mathbb{C}^{\mathsf{tm}}_\mathsf{F}(L, x\,\mathsf{ty})}\ x \notin L \qquad \frac{\mathbb{C}^{\mathsf{tm}}_\mathsf{F}(L) \qquad \{L \vdash A\,\mathsf{ty}\}}{\mathbb{C}^{\mathsf{tm}}_\mathsf{F}(L, x :_\mathsf{F} A)}\ x \notin L, A$$

The freshness conditions are implemented by locally ∇-quantifying the respective variable. Next we have to pair this definition with an inversion principle that reveals the structure and freshness properties of any $J \in L$ with $\mathbb{C}^{\mathsf{tm}}_\mathsf{F}(L)$. At this point we are able to add $\mathbb{C}^{\mathsf{ty}}_\mathsf{F}(L)$, defined analogously to $\mathbb{C}^{\mathsf{tm}}_\mathsf{F}(L)$, as an extra premise to our inversion principle and then discard the spurious context extraction. This relies on the fact that $\nabla x.(A \to B) \neq x$ holds in $\mathcal{G}$. A key aspect of formalising our equivalence result in Abella is the correct choice of well-formedness predicates and associated inversion lemmas.

[2] The HOAS language definitions are standard. See also Appendix B for reference.

The second equivalence proof. The relations $\sim$ and $\approx$ are defined as:

$$\frac{A \sim a \quad \mathbf{\Pi} x.\ B \sim b\langle x\rangle}{A \to B \sim \Pi a.\ b} \qquad \frac{\mathbf{\Pi} xy.\ x \sim y \Rightarrow A\langle x\rangle \sim a\langle y\rangle}{\forall.\ A \sim \Pi *.\ a} \qquad \frac{s \approx a \quad t \approx b}{s\,t \approx a\,b}$$

$$\frac{A \sim a \quad \mathbf{\Pi} xy.\ x \approx y \Rightarrow s\langle x\rangle \approx b\langle y\rangle}{\lambda A.\ s \approx \lambda a.\ b} \qquad \frac{\mathbf{\Pi} xy.\ x \sim y \Rightarrow s\langle x\rangle \approx a\langle y\rangle}{\Lambda.\ s \approx \lambda *.\ a} \qquad \frac{s \approx a \quad A \sim b}{s\,A \approx a\,b}$$

We complement this definition with a well-formedness predicate $\mathbb{C}_{\approx}(L)$, which ascertains that L only contains judgements of the form $n_i \sim n_j$ or $n_i \approx n_j$, together with suitable lookup lemmas. For technical reasons we do not define $\mathbb{C}_{\sim}(-)$ and instead prove a strengthening lemma which holds due to the inferred subordination ordering of the two relations:

$$\mathbb{C}_{\approx}(L) \implies \{L, x \approx y \vdash A \sim a\} \implies \{L \vdash A \sim a\}$$

Before we go on, it is interesting to consider the information encapsulated in $\mathbb{C}_{\approx}(L)$. The obvious part is that L contains exactly the same information about corresponding variables that we had to track in Coq with the auxiliary parameters R and S. In addition, since L only contains pairings of fresh nominals, we immediately obtain that L is functional and injective. Lifting these properties to $\sim$ and then to $\approx$ are routine inductions.

When it comes to the totality and preservation statements, things become more interesting. The remaining four lemma statements are generalised over three different contexts belonging to the three involved judgements. Not only do we require these contexts to be locally well-formed, but we also have to connect them. We achieve this with a single ternary well-formedness predicate, $\mathbb{C}_{\mathsf{R}}(- \mid - \mid -)$, defined as follows:

$$\frac{}{\mathbb{C}_{\mathsf{R}}(\bullet \mid \bullet \mid \bullet)} \qquad \frac{\mathbb{C}_{\mathsf{R}}(L_{\mathsf{F}} \mid L_{\approx} \mid L_{\lambda}) \quad x, y \notin L_i}{\mathbb{C}_{\mathsf{R}}(L_{\mathsf{F}}, x\,\mathsf{ty} \mid L_{\approx}, x \sim y \mid L_{\lambda}, y :_{\lambda} *)} \qquad \frac{\mathbb{C}_{\mathsf{R}}(L_{\mathsf{F}} \mid L_{\approx} \mid L_{\lambda}) \quad x, y \notin L_i, A, a \quad \{L_{\mathsf{F}} \vdash A\,\mathsf{ty}\} \quad \{L_{\approx} \vdash A \sim a\} \quad \{L_{\lambda} \vdash a :_{\lambda} *\}}{\mathbb{C}_{\mathsf{R}}(L_{\mathsf{F}}, x :_{\mathsf{F}} A \mid L_{\approx}, x \approx y \mid L_{\lambda}, y :_{\lambda} a)}$$

When a binder is traversed, this ensures that all three contexts are extended with the same freshly chosen variables x and y. The definition is accompanied by three extraction lemmas, one for each of the three involved contexts, which provide the associated judgements from the two other contexts. Recall that in Coq we had four separate invariants for the four preservation and totality lemmas. Here, this information is uniformly encoded in $\mathbb{C}_{\mathsf{R}}(L_{\mathsf{F}} \mid L_{\approx} \mid L_{\lambda})$ and we use it for all four proofs.

▶ **Lemma 18.** *The type relation $\sim$ is total from F to $\lambda 2$ and preserves type formation.*

$$\{L_{\mathsf{F}} \vdash A\,\mathsf{ty}\} \implies \forall L_{\approx} L_{\lambda}.\ \mathbb{C}_{\mathsf{R}}(L_{\mathsf{F}} \mid L_{\approx} \mid L_{\lambda}) \implies \exists a.\ \{L_{\approx} \vdash A \sim a\} \wedge \{L_{\lambda} \vdash a :_{\lambda} *\}$$

Proof. By induction on $\{L_F \vdash A\,\mathsf{ty}\}$. ◀

As a trivial corollary we obtain: $\{A\,\mathsf{ty}\} \implies \exists a.\ \{A \sim a\} \wedge \{a :_{\lambda} *\}$.

The proofs of the remaining three preservation results are surprisingly analogue, so we will not go into further detail. It is worth pointing out, though, that we again require the degeneracy of the PTS universe $\Box$, as well as propagation for both F and $\lambda 2$ in various places. The HOAS setup allows us to obtain these easily.

5 Beluga

The programming and proof environment Beluga [24, 25] is another system that supports HOAS. Object languages are encoded in the logical framework LF [18], while proofs about these are expressed as total programs in contextual modal type theory, Beluga's reasoning logic. The programs analyse LF derivation trees using pattern matching and higher-order unification. In Beluga, it is necessary to give programs (i.e. proof terms) explicitly, which are proof checked as part of type checking. This stands in contrast to Coq and Abella, both of which are interactive systems with a tactic language for proof term construction.

The biggest difference, however, lies with Beluga's treatment of object level variables. In particular, at the outermost level, there is no such thing as a free object variable. This is in stark contrast to Coq's dangling de Bruijn indices and Abella's global nominal constants. In Beluga we instead deal with *contextual objects*, written $[\,\Gamma \vdash K\,]$, that is objects K (like types, terms, typing derivations) paired with contexts Γ in which they are meaningful [20, 23]. One of the main advantages of contextual objects is that they remain closed under inductive reasoning. Hence they constitute an alternative to Abella's two-level logical embedding.

As an example, consider the function type $X \to X$ where X is a free type variable. This function type is ill-formed under the empty type variable context $\Delta = \emptyset$. In Coq and Abella we can express this type, as $0_{\mathsf{ty}} \to 0_{\mathsf{ty}}$ and respectively $n_0 \to n_0$. We can then show that assuming well-formedness under the empty context entails absurdity, $0 \mid\!\!\!\stackrel{\mathsf{ty}}{\mathsf{F}}\ 0_{\mathsf{ty}} \to 0_{\mathsf{ty}} \implies \bot$ and $\{\bullet \vdash n_0 \to n_0\,\mathsf{ty}\} \implies \bot$. Meanwhile in Beluga, we observe that the contextual object $[\,\bullet \vdash x \to x\,]$ is not even syntactically well-formed, since $x \notin \bullet$. An immediate consequence of this is that the type formation judgement for F, which ensures that the context is covering all free type variables, becomes redundant. Hence Beluga's definition of F is obtained from Figure 6 by removing all references to type formation. The definition of λ2 is identical to the one for Abella (Figure 7).[3]

Recall that context management was completely manual in Coq. Each judgement required well-chosen generalisations and custom invariants to accurately track contextual information. In Abella the situation was noticeably better, as contexts at the object level were kept implicit and handled by the system. At the reasoning level they did, however, surface as explicit, unstructured sequences of judgements. The desired contextual structure then had to be imposed with auxiliary predicates, together with copious amounts of inversion lemmas.

Beluga contexts, on the other hand, are sequences of not necessarily homogeneous declarations. Each declaration can depend on prior declarations and encapsulate multiple pieces of related information using a dependent record. Contexts are first class citizens and *context schema* ascription, $\Gamma : S$, is used to ensure that a given context Γ satisfies certain structural constraints.

Schemas are Beluga's main device to enforce invariants on contextual information. We use propagation for λ2 as an example. The requisite schema is:

$$S_{\lambda W} := [x{:}\,\mathsf{Tm}_\lambda,\ x :_\lambda *] + [x{:}\,\mathsf{Tm}_\lambda,\ x :_\lambda a,\ a :_\lambda *]$$

Note how it separates PTS variables into type and term variables via the associated typing information, which already imposes the necessary semantic contextual information. The proof of propagation is straightforward. We implement a total recursive function k

[3] For a concrete presentation of the resulting definitions, see Appendix B.

satisfying:

$$k \,:\, \forall \Gamma{:}S_{\lambda W}.\, [\,\Gamma \vdash a :_\lambda b\,] \implies [\,\Gamma \vdash \mathsf{type\_correct}\, b\,]$$

The contextual predicate $[\,\Gamma \vdash \mathsf{type\_correct}\, b\,]$ encodes that b is either $\Box$ or it can be typed with some universe u. We pattern match $\mathcal{D} : [\,\Gamma \vdash a :_\lambda b\,]$ and obtain seven cases. The first four are structural, recursively descending into sub-derivations. The only part that is non-obvious is the traversal of binders, where various pieces of information are added to the context. These have to be packaged into declaration blocks in order to satisfy $S_{\lambda W}$ for the recursive call. More interesting though are the three base cases. When we compare the matched type against $S_{\lambda W}$, we observe three ways in which $\mathcal{D}$ could have been obtained from the context. The first and third are trivial as they unify b with $*$ which can be typed with the universe $\Box$. For the remaining case we know that b can be typed with the universe $*$, contextual information that was packaged together with the matched judgement. Note that throughout the construction we exploit that Beluga natively supports substitution into parametric sub-derivations.

The third equivalence proof. The definitions of $\sim$ and $\approx$ exactly coincide with Abella. We are going to primarily concern ourselves with the schemas, which best illustrate Beluga's tracking of contextual information.

We begin with the functionality and injectivity properties of $\sim$ and $\approx$. Since equality is not native in Beluga we have to define equality predicates for each syntactic sort to express our statements. The tightest invariants that hold for the rules defining $\sim$ and $\approx$ can be expressed with the following context schemas:

$$S_\sim := [x{:}\,\mathsf{Ty_F},\, y{:}\,\mathsf{Tm}_\lambda,\, x \sim y] + [y{:}\,\mathsf{Tm}_\lambda]$$
$$S_\approx := [x{:}\,\mathsf{Ty_F},\, y{:}\,\mathsf{Tm}_\lambda,\, x \sim y] + [x{:}\,\mathsf{Tm_F},\, y{:}\,\mathsf{Tm}_\lambda,\, x \approx y]$$

Due to the subordination ordering of $\sim$ and $\approx$, Beluga is capable of automatically strengthening from $S_\approx$ to $S_\sim$, and weaken vice versa (see also [29]).

▶ **Lemma 19.** *There exist total recursive functions $f_{\mathsf{ty}}, f_{\mathsf{tm}}, i_{\mathsf{ty}}$ and i_{tm}, satisfying*

$$f_{\mathsf{ty}} \,:\, \forall \Gamma{:}S_\sim.\, [\,\Gamma \vdash A \sim a\,] \implies [\,\Gamma \vdash A \sim a'\,] \implies [\,\Gamma \vdash a =_\lambda a'\,]$$
$$i_{\mathsf{ty}} \,:\, \forall \Gamma{:}S_\sim.\, [\,\Gamma \vdash A \sim a\,] \implies [\,\Gamma \vdash A' \sim a\,] \implies [\,\Gamma \vdash A =^{\mathsf{ty}}_{\mathsf{F}} A'\,]$$
$$f_{\mathsf{tm}} \,:\, \forall \Gamma{:}S_\approx.\, [\,\Gamma \vdash s \approx a\,] \implies [\,\Gamma \vdash s \approx a'\,] \implies [\,\Gamma \vdash a =_\lambda a'\,]$$
$$i_{\mathsf{tm}} \,:\, \forall \Gamma{:}S_\approx.\, [\,\Gamma \vdash s \approx a\,] \implies [\,\Gamma \vdash s' \approx a\,] \implies [\,\Gamma \vdash s =^{\mathsf{tm}}_{\mathsf{F}} s'\,]$$

Proof. Each by induction on the first premise and pattern matching on the second. In the variable cases we have matched against two context records r and r'. Since x and y are local to r and one of them is shared between the two matched records, unification infers $r = r'$, closing the case. Note that f_{tm} and i_{tm} contain calls to f_{ty} and respectively i_{ty}, which relies on context strenghtening. For i_{tm} we also again require disjointedness of the two relations, which is easily obtainable under contexts satisfying $S_\approx$. ◀

For the remaining four totality and preservation proofs we have to deal with two complications. First, we have to remove assumptions and conclusions referring to F type formation. Second, we have to define predicates that capture the existential nature of the four conclusions, including relevant typing information, with custom predicates. We have, for example,

$$\mathsf{exists\_rel\_proof}\, s\, A \iff \exists ba.\; s \approx b \;\wedge\; A \sim a \;\wedge\; b :_\lambda a \;\wedge\; a :_\lambda *$$

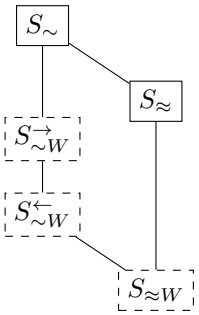


Figure 8 Hierarchy of Context Schemas.

The required context schemas are quite involved:

$$S^{\rightarrow}_{\sim W} := [x\colon \mathsf{Ty_F},\ y\colon \mathsf{Tm}_\lambda,\ x \sim y,\ y :_\lambda *] + [y\colon \mathsf{Tm}_\lambda,\ y :_\lambda a]$$
$$S^{\leftarrow}_{\sim W} := [x\colon \mathsf{Ty_F},\ y\colon \mathsf{Tm}_\lambda,\ x \sim y,\ y :_\lambda *] + [y\colon \mathsf{Tm}_\lambda,\ y :_\lambda a,\ A \sim a,\ a :_\lambda *]$$
$$S_{\approx W} := [x\colon \mathsf{Ty_F},\ y\colon \mathsf{Tm}_\lambda,\ x \sim y,\ y :_\lambda *] + [x\colon \mathsf{Tm_F},\ y\colon \mathsf{Tm}_\lambda,\ x \approx y,\ x :_\mathsf{F} A,\ y :_\lambda a,\ A \sim a]$$

▶ **Lemma 20.** *There exist total recursive functions* $p^{\rightarrow}_{\sim}, p^{\leftarrow}_{\sim}, p^{\rightarrow}_{\approx}$ *and* $p^{\leftarrow}_{\approx}$*, satisfying*

$$p^{\rightarrow}_{\sim} : \forall \Gamma : S^{\rightarrow}_{\sim W}.\ \forall A : [\Gamma \vdash \mathsf{Ty_F}].\ [\Gamma \vdash \mathsf{exists\_rel\_prop}\ A]$$
$$p^{\leftarrow}_{\sim} : \forall \Gamma : S^{\leftarrow}_{\sim W}.\ [\Gamma \vdash a :_\lambda *] \implies [\Gamma \vdash \mathsf{exists\_rel\_type}\ a]$$
$$p^{\rightarrow}_{\approx} : \forall \Gamma : S_{\approx W}.\ [\Gamma \vdash s :_\mathsf{F} A] \implies [\Gamma \vdash \mathsf{exists\_rel\_proof}\ s\ A]$$
$$p^{\leftarrow}_{\approx} : \forall \Gamma : S_{\approx W}.\ [\Gamma \vdash b :_\lambda a] \implies [\Gamma \vdash a :_\lambda *] \implies [\Gamma \vdash \mathsf{exists\_rel\_term}\ b\ a]$$

Proof. The first is by induction on $A : [\Gamma \vdash \mathsf{Ty_F}]$ (recall that this was not possible in Abella), the others are by induction on the first premise. The proofs are quite technical but mostly straightforward. The construction of $p^{\leftarrow}_{\approx}$, needs λ2 propagation. Interestingly, neither propagation in F nor the degeneracy of □ are needed, as unification automatically handles the respective occurrences. ◀

The most interesting part of the Beluga development appears to be the particularly rich structure and interdependencies of the various schemas. We would like to point out in particular, that while the schemas $S_\sim$ and $S_\approx$ could likely be inferred automatically by inspecting the involved type families, this does not appear to work for those schemas with auxiliary well-typedness assumptions (subscript W). This contradicts the common belief that schema inference should in principle always be possible.

The schemas can be further arranged in a hierarchy (Figure 8). A context satisfying $S_\sim$ can always be weakened to one sitting lower in the hierarchy. The hierarchy also induces a strengthening relationship, going upwards, as long as the subordination order of judgements under said contexts is respected.

6 Adequacy

To put any trust in the formalisations we have just discussed, we have to briefly consider the issue of *adequacy*. That is, we have to argue why our formal definitions correspond to our intuitive understanding of the mathematical objects at hand.

For the de Bruijn setup in Coq, the question does not really arise, as it is a well understood first-order encoding. The representation of the syntax is comparable to any other inductive

datatype, like say natural numbers or lists. It is our understanding that a de Bruijn encoding is the canonical implementation of the *Barendregt convention*. The fact that all three proof assistants used here are internally implemented using de Bruijn supports this belief.

The situation is quite different for our HOAS encodings, as they borrow their function spaces and substitution mechanisms directly from their host environment. When HOAS was first introduced it was not at all clear that this would yield sensible syntactic structures. Thus since at least the 1990s a lot of techniques were developed to argue that such definitions are faithful. We rely on these to trust our language definitions.

When it comes to the HOAS type systems and proofs about them, the situation becomes less clear, as both Abella and Beluga go beyond basic λ-tree syntax and exploit subordination to justify the inductiveness of certain proofs. Here we can argue adequacy based on two facts. First, we have proven that in each case two variants of intuitively the same mathematical system do in fact behave the same and the encodings also admit all the expected properties. Second, we were able to replay the same overall proof structure that worked for the de Bruijn approach on the HOAS encodings. Taken together, this stability within and across proof systems allows us to assume the adequacy of representations until evidence to the contrary is provided.

7 Conclusion

We have considered a technically interesting proof and demonstrated how various formalisation techniques deal with the arising intricacies. The development of three different formalisations allowed us to gain deep insights into the inherent complexities of the proof. In particular we were able to separate these from technical artefacts due to the chosen formalisation technique. Two examples of inherent complications are the not quite perfectly aligned binding structures and the missing typing information required to disambiguate the uniform PTS applications.

Our set of developments demonstrates that the various formalisation techniques can be arranged in a hierarchy of abstraction layers. At the lowest level we have pure de Bruijn with a lot of representation freedom, which, however, has to be manged manually. Higher up in the hierarchy sit the HOAS techniques, which hide a lot of the technicalities and provide a more meaningful abstraction. In comparison with Abella, Beluga appears to deliver the theoretically nicer interface, with the added features of contextual reasoning and the ability to perform inductions directly over the HOAS syntax. Practically though, both systems are relatively young with certain usability issues. Among the two, Abella's tactic language certainly gives it a head start.

We observe that our proof contains a number of challenges that, taken as a whole, constitute a nice benchmark for systems designed to reason about type systems. It tests in particular, how well multiple type systems with binding constructs can be brought into correspondence. We hence propose it as a complement to the POPLmark challenge [2] on the one hand, which solely tests how well a framework can reason about a single type system, and the ORBI benchmarks [10, 9] which cover small-scale contextual reasoning. It is our belief that such a benchmark could be very useful to those who seek to develop or improve frameworks for reasoning about type systems and similar syntactic systems with variable binding and complex contextual information.

We have come to the conclusion that there appears to be no "silver bullet" solution. There exist various approaches and they all have their merits and drawbacks. The question is not so much about whether de Bruijn or HOAS is better, but what techniques pair well with each approach. We have highlighted some of these here but there are of course further

approaches (see below). A comprehensive survey, while desirable, would certainly exceed the scope of this format, but we hope that our work constitutes a small step towards such a catalogue of techniques. To conclude, we found that reworking the same proof in multiple frameworks was quite enlightening.

Future Work. We would like to continue into three largely orthogonal directions.

First, we would like to widen the scope of the benchmark itself and include a correspondence result for the computational behaviour of the two systems. Equi-reducebility is at least as interesting a problem as equi-typability, and likely to pose its own set of challenges. One of these is the fact that, prior to typing, $\lambda 2$ has a lot more β-redices than its two-sorted counterpart, hence some form of typing information will have to be tracked along with the relational assumptions. This will lead to new forms of contexts that are likely comparable in complexity to those discussed here.

Second, we would like to test further frameworks against our benchmark. One candidate is the locally nameless approach [3]. It provides an abstraction layer that sits somewhere between pure de Bruijn and HOAS. Our benchmark could test the stability of this layer. Another is the HYBRID framework [8] that aims to bring HOAS to Coq. At present however, it is neither equipped with Abella's ∇ nor Beluga's contextual types, so it remains to be seen if it can handle our challenge. Note that the HYBRID library exists in Isabelle as well. The rich context structures could also pose an interesting challenge for systems like Twelf [22].

Finally, we consider scaling the challenge both down to the simply typed λ-calculus, as well as up to F_ω, to obtain a better understanding of where exactly certain complications originate from.

References

1 Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *JFP*, 1(4):375–416, 1991.

2 Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic. Mechanized metatheory for the masses: The POPLmark challenge. In *TPHOL*, LNCS 3603, pages 50–65. Springer, 2005.

3 Brian E. Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. Engineering formal metatheory. In *POPL 2008*, pages 3–15. ACM, 2008.

4 David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu, and Yuting Wang. Abella: A system for reasoning about relational specifications. *JFR*, 7(2):1–89, 2014.

5 Henk Barendregt. Introduction to generalized type systems. *JFP*, 1(2):125–154, 1991.

6 The Coq proof assistant. URL: `http://coq.inria.fr/`.

7 Nicolaas Govert de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972.

8 Amy P. Felty and Alberto Momigliano. Hybrid – A definitional two-level approach to reasoning with higher-order abstract syntax. *JAR*, 48(1):43–105, 2012.

9 Amy P. Felty, Alberto Momigliano, and Brigitte Pientka. The next 700 challenge problems for reasoning with higher-order abstract syntax representations: Part 2 – a survey. *Journal of Automated Reasoning*, 55(4):307–372, 2015.

10 Amy P. Felty and Brigitte Pientka. Reasoning with higher-order abstract syntax and contexts: A comparison. In *ITP 2010*, LNCS 6172, pages 227–242. Springer, 2010.

11 Andrew Gacek, Dale Miller, and Gopalan Nadathur. Combining generic judgments with recursive definitions. In Frank Pfenning, editor, *LICS 2008*. IEEE Computer Society Press, 2008.
12 Andrew Gacek, Dale Miller, and Gopalan Nadathur. A two-level logic approach to reasoning about computations. *JAR*, 49(2):241–273, 2012.
13 Jan Herman Geuvers. Logics and type systems. Proefschrift, Katholieke Univ. Nijmegen, 1993.
14 Jean-Yves Girard. Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur. Thèse de doctorat d'état, Université Paris VII, 1972.
15 Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*. Cambridge Univ. Press, 1989.
16 Healfdene Goguen and James McKinna. Candidates for substitution. Technical Report ECS-LFCS-97-358, Univ. of Edinburgh, 1997.
17 Robert Harper. *Practical foundations for programming languages*. Cambridge Univ. Press, 2013.
18 Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, January 1993.
19 Jonas Kaiser, Tobias Tebbi, and Gert Smolka. Equivalence of System F and $\lambda 2$ in Coq based on context morphism lemmas. In *CPP 2017*. ACM, 2017.
20 Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):1–49, 2008.
21 Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *PLDI 1988*, pages 199–208. ACM, 1988.
22 Frank Pfenning and Carsten Schürmann. System description: Twelf – A meta-logical framework for deductive systems. In *CADE 1999*, pages 202–206. Springer, 1999.
23 Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *POPL 2008*, pages 371–382. ACM Press, 2008.
24 Brigitte Pientka. Beluga: programming with dependent types, contextual data, and contexts. In *FLOPS 2010*, LNCS 6009, pages 1–12. Springer, 2010.
25 Brigitte Pientka and Andrew Cave. Inductive Beluga:Programming Proofs (System Description). In *CADE 2015*, LNCS 9195, pages 272–281. Springer, 2015.
26 John Charles Reynolds. Towards a theory of type structure. In *Paris Colloque sur la Programmation*, LNCS 19, pages 408–423. Springer, 1974.
27 Steven Schäfer, Gert Smolka, and Tobias Tebbi. Completeness and decidability of de Bruijn substitution algebra in Coq. In *CPP 2015*, pages 67–73. ACM, 2015.
28 Steven Schäfer, Tobias Tebbi, and Gert Smolka. Autosubst: Reasoning with de Bruijn terms and parallel substitutions. In *ITP 2015*, LNCS 9236, pages 359–374. Springer, 2015.
29 Roberto Virga. *Higher-Order Rewriting with Dependent Types*. PhD thesis, Dep. of Math. Sciences, Carnegie Mellon Univ., 1999. CMU-CS-99-167.

A De Bruijn Definitions

We summarise the core concepts of de Bruijn with parallel substitutions. The main feature of de Bruijn syntax is the absence of variable names, yielding concrete canonical representations for the on-paper implicitly assumed α-equivalence classes of syntactic expressions. Instead of using names, variables are represented as numerical indices, where n references the n-th enclosing binder of the corresponding scope, counting from 0. We illustrate the concepts using the two-sorted variant of System F that is discussed in the main text:

$$A, B ::= x_{\mathsf{ty}} \mid A \to B \mid \forall.\, A \qquad s, t ::= x_{\mathsf{tm}} \mid s\, t \mid \lambda A.\, s \mid s\, A \mid \Lambda.\, s \qquad x : \mathbb{N}$$

Note that the A in $\lambda A.\, s$ is the type of the bound term variable, not its name. Dangling indices correspond to free variables. They are taken as indices into an ambient context which is normally represented as a list of the information attached to each variable. A standard typing context Γ is represented as a simple list of types.

To understand the instantiation of terms or types with parallel substitutions, let us first recall a few primitives of the σ-calculus (see [1] for details):

$$\begin{aligned} (M \cdot \sigma)\, 0 &:= M & \uparrow x &:= (x+1) \\ (M \cdot \sigma)\, (x+1) &:= \sigma\, x & \mathsf{id} &:= 0 \cdot \uparrow \end{aligned}$$

Parallel substitutions $\sigma : \mathbb{N} \to \mathcal{T}$, where $\mathcal{T}$ is some syntactic sort, can be seen as streams $M_0, M_1, \ldots$, with $M_i : \mathcal{T}$. This motivates the notion of the *cons* operation $M \cdot \sigma$, which maps the index 0 to M, and all other indices $x+1$ to $\sigma\, x$. The *shift* operation $\uparrow$ simply raises all indices by one, and the identity substitution id is a derived notion. The concrete type of substitutions and a corresponding instance of this framework depends on the syntactic sort for which a notion of substitution is defined.

Let us first consider the types of our example. The instantiation of a type A with a parallel substitution σ, written $A[\sigma]$, is defined mutually recursive with the forward composition of parallel substitutions.

$$\begin{aligned} x_{\mathsf{ty}}[\sigma] &= \sigma\, x_{\mathsf{ty}} & (\sigma_1 \circ \sigma_2)\, x_{\mathsf{ty}} &= (\sigma_1\, x_{\mathsf{ty}})[\sigma_2] \\ (A \to B)[\sigma] &= A[\sigma] \to B[\sigma] & & \\ (\forall.\, A)[\sigma] &= \forall.\, A[\Uparrow \sigma] & \text{with } \Uparrow \sigma &= 0_{\mathsf{ty}} \cdot \sigma \circ \uparrow \end{aligned}$$

For the terms we actually require a vector of two parallel substitutions, $\langle \sigma, \tau \rangle$, since both type and term variables can occur in a given term:

$$\begin{aligned} x_{\mathsf{tm}}[\sigma, \tau] &= \tau\, x_{\mathsf{tm}} & (\sigma' \circ \langle \sigma, \tau \rangle)\, x_{\mathsf{tm}} &= (\sigma'\, x_{\mathsf{tm}})[\sigma, \tau] \\ (s\, t)[\sigma, \tau] &= s[\sigma, \tau]\, t[\sigma, \tau] & & \\ (\lambda A.\, s)[\sigma, \tau] &= \lambda A[\sigma].\, s[\Uparrow^{\mathsf{tm}} \langle \sigma, \tau \rangle] & \text{with } \Uparrow^{\mathsf{tm}} \langle \sigma, \tau \rangle &= \langle \sigma, 0_{\mathsf{tm}} \cdot \tau \circ \langle \mathsf{id}, \uparrow \rangle \rangle \\ (s\, A)[\sigma, \tau] &= s[\sigma, \tau]\, A[\sigma] & & \\ (\Lambda.\, s)[\sigma, \tau] &= \Lambda.\, s[\Uparrow^{\mathsf{ty}} \langle \sigma, \tau \rangle] & \text{with } \Uparrow^{\mathsf{ty}} \langle \sigma, \tau \rangle &= \langle 0_{\mathsf{ty}} \cdot \sigma \circ \uparrow, \tau \circ \langle \uparrow, \mathsf{id} \rangle \rangle \end{aligned}$$

The key idea in both cases is that parallel substitutions act on all free variables at once, which leads to an elegant equational theory and subsequently to a good foundation for proof automation. A nice example is β-reduction, which can concisely be expressed as $(\lambda A.\, s)\, t \rightsquigarrow s[\mathsf{id}, t \cdot \mathsf{id}]$.

The Autosubst framework is capable of generating all of this automatically from annotated inductive syntax definitions, together with the respective equational theories. It further provides a decision tactic that can solve goals involving substitution expressions, based on the results in [27].

B HOAS Definitions

We give a brief overview of the definitions as they were used in the Abella and Beluga proofs. The HOAS specifications are identical for both systems. The two-sorted variant is given on the left, the PTS is on the right. We also include the types of the judgements.

$$\mathsf{Ty_F}, \mathsf{Tm_F} : \mathbf{Type} \qquad \mathsf{Tm}_\lambda : \mathbf{Type}$$

$$\begin{array}{rl|rl} & & *, \square & : \mathsf{Tm}_\lambda \\ \_ \to \_ & : \mathsf{Ty_F} \to \mathsf{Ty_F} \to \mathsf{Ty_F} & \Pi \_. \_ & : \mathsf{Tm}_\lambda \to (\mathsf{Tm}_\lambda \to \mathsf{Tm}_\lambda) \to \mathsf{Tm}_\lambda \\ \forall. \_ & : (\mathsf{Ty_F} \to \mathsf{Ty_F}) \to \mathsf{Ty_F} & & \\ \_\,\_ & : \mathsf{Tm_F} \to \mathsf{Tm_F} \to \mathsf{Tm_F} & \_\,\_ & : \mathsf{Tm}_\lambda \to \mathsf{Tm}_\lambda \to \mathsf{Tm}_\lambda \\ \_\,\_ & : \mathsf{Tm_F} \to \mathsf{Ty_F} \to \mathsf{Tm_F} & & \\ \lambda \_. \_ & : \mathsf{Ty_F} \to (\mathsf{Tm_F} \to \mathsf{Tm_F}) \to \mathsf{Tm_F} & \lambda \_. \_ & : \mathsf{Tm}_\lambda \to (\mathsf{Tm}_\lambda \to \mathsf{Tm}_\lambda) \to \mathsf{Tm}_\lambda \\ \Lambda. \_ & : (\mathsf{Ty_F} \to \mathsf{Tm_F}) \to \mathsf{Tm_F} & & \end{array}$$

$$\begin{array}{rl|rl} \_\,\mathsf{ty} & : \mathsf{Ty_F} \to \mathbf{Prop} & \mathcal{U}\ \_ & : \mathsf{Tm}_\lambda \to \mathbf{Prop} \\ \_ :_\mathsf{F} \_ & : \mathsf{Tm_F} \to \mathsf{Ty_F} \to \mathbf{Prop} & \_ :_\lambda \_ & : \mathsf{Tm}_\lambda \to \mathsf{Tm}_\lambda \to \mathbf{Prop} \end{array}$$

Note that Abella distinguishes the system sorts **Prop** and **Type**. Inductive derivations are only exposed for the former. In Beluga on the other hand, both are considered as basic LF types.

The four judgements/predicates are defined inductively, both in Abella and Beluga. The two PTS definitions are identical in both frameworks. The bold operators $\Rightarrow$ and $\mathbf{\Pi}\_.\ \_$ are used to construct hypothetical and locally quantified premises, respectively. Meta level application is denoted by $\_\langle\_\rangle$.

$$\frac{}{\mathcal{U}\,\square} \qquad \frac{}{\mathcal{U}\,*} \qquad \frac{}{* :_\lambda \square} \qquad \frac{a :_\lambda \Pi c.\, d \qquad b :_\lambda c}{a\,b :_\lambda d\langle b\rangle} \qquad \frac{a :_\lambda u \qquad \mathcal{U}\,u \qquad \mathbf{\Pi} x.\ x :_\lambda a \Rightarrow b\langle x\rangle :_\lambda *}{\Pi a.\, b :_\lambda *}$$

$$\frac{a :_\lambda u \qquad \mathcal{U}\,u \qquad \mathbf{\Pi} x.\ x :_\lambda a \Rightarrow c\langle x\rangle :_\lambda * \qquad \mathbf{\Pi} x.\ x :_\lambda a \Rightarrow b\langle x\rangle :_\lambda c\langle x\rangle}{\lambda a.\, b :_\lambda \Pi a.\, c}$$

With respect to the two-sorted system there is a difference between the Abella and the Beluga definitions, due to the differences in the respective meta theories. The Abella version is straightforward:

$$\frac{A\,\mathsf{ty} \qquad B\,\mathsf{ty}}{(A \to B)\,\mathsf{ty}} \qquad \frac{\mathbf{\Pi} x.\ x\,\mathsf{ty} \Rightarrow A\langle x\rangle\,\mathsf{ty}}{(\forall.\, A)\,\mathsf{ty}} \qquad \frac{s :_\mathsf{F} \forall.\, B \qquad A\,\mathsf{ty}}{s\,A :_\mathsf{F} B\langle A\rangle} \qquad \frac{s :_\mathsf{F} A \to B \qquad t :_\mathsf{F} A}{s\,t :_\mathsf{F} B}$$

$$\frac{\mathbf{\Pi} x.\ x\,\mathsf{ty} \Rightarrow s\langle x\rangle :_\mathsf{F} A\langle x\rangle}{\Lambda.\, s :_\mathsf{F} \forall.\, A} \qquad \frac{A\,\mathsf{ty} \qquad \mathbf{\Pi} x.\ x :_\mathsf{F} A \Rightarrow s\langle x\rangle :_\mathsf{F} B}{\lambda A.\, s :_\mathsf{F} A \to B}$$

In Beluga, on the other hand, contexts are first class, and thus well-scopedness of expressions is inherent. Hence *the type formation judgement of the two-sorted system vanishes* completely, which reduces the definition of the typing judgement to:

$$\frac{s :_\mathsf{F} \forall.\, B}{s\,A :_\mathsf{F} B\langle A\rangle} \qquad \frac{s :_\mathsf{F} A \to B \qquad t :_\mathsf{F} A}{s\,t :_\mathsf{F} B} \qquad \frac{\mathbf{\Pi} x.\ s\langle x\rangle :_\mathsf{F} A\langle x\rangle}{\Lambda.\, s :_\mathsf{F} \forall.\, A} \qquad \frac{\mathbf{\Pi} x.\ x :_\mathsf{F} A \Rightarrow s\langle x\rangle :_\mathsf{F} B}{\lambda A.\, s :_\mathsf{F} A \to B}$$

A Polynomial-Time Algorithm for the Lambek Calculus with Brackets of Bounded Order*†

Max Kanovich[1], Stepan Kuznetsov[2], Glyn Morrill[3], and Andre Scedrov[4]

1 **National Research University Higher School of Economics, Moscow, Russia**
mkanovich@hse.ru
2 **Steklov Mathematical Institute of RAS, Moscow, Russia; and**
National Research University Higher School of Economics, Moscow, Russia
sk@mi.ras.ru
3 **Universitat Politècnica de Catalunya, Barcelona, Spain**
morrill@cs.upc.edu
4 **University of Pennsylvania, Philadelphia, USA; and**
National Research University Higher School of Economics, Moscow, Russia
scedrov@math.upenn.edu

Abstract

Lambek calculus is a logical foundation of categorial grammar, a linguistic paradigm of grammar as logic and parsing as deduction. Pentus (2010) gave a polynomial-time algorithm for determining provability of bounded depth formulas in $\mathbf{L}^*$, the Lambek calculus with empty antecedents allowed. Pentus' algorithm is based on tabularisation of proof nets. Lambek calculus with brackets is a conservative extension of Lambek calculus with bracket modalities, suitable for the modeling of syntactical domains. In this paper we give an algorithm for provability in $\mathbf{Lb}^*$, the Lambek calculus with brackets allowing empty antecedents. Our algorithm runs in polynomial time when both the formula depth and the bracket nesting depth are bounded. It combines a Pentus-style tabularisation of proof nets with an automata-theoretic treatment of bracketing.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases Lambek calculus, proof nets, Lambek calculus with brackets, categorial grammar, polynomial algorithm

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.22

1 Introduction

The calculus $\mathbf{L}$ of Lambek [23] is a logic of strings. It is retrospectively recognisable as the multiplicative fragment of non-commutative intuitionistic linear logic without empty antecedents; the calculus $\mathbf{L}^*$ is like $\mathbf{L}$ except that it admits empty antecedents. The Lambek calculus is the foundation of categorial grammar, a linguistic paradigm of grammar as

* The work of M. Kanovich and A. Scedrov was supported by the Russian Science Foundation under grant 17-11-01294 and performed at National Research University Higher School of Economics, Russia. The work of G. Morrill was supported by an ICREA Academia 2012 and MINECO TIN2014-57226-P (APCOM). The work of S. Kuznetsov was supported by the Russian Foundation for Basic Research (grant 15-01-09218-a) and by the Presidential Council for Support of Leading Research Schools (grant НШ-9091.2016.1).

† Section 1 was contributed by G. Morrill, Section 4 by M. Kanovich and A. Scedrov, Section 5 by S. Kuznetsov, and Sections 2, 3, 6, and 7 were contributed jointly and equally by all coauthors.

logic and parsing as deduction; see for instance Buszkowski [6], Carpenter [7], Jäger [17], Morrill [28], Moot and Retoré [26]. For example, the sentence "*John knows Mary likes Bill*" can be analysed as grammatical because $N, (N \setminus S) / S, N, (N \setminus S) / N, N \to S$ is a theorem of Lambek calculus. Here N stands for *noun phrase,* S stands for *sentence,* and syntactic categories for other words are built from these two primitive ones using division operations. For example, $(N \setminus S) / N$ takes noun phrases on both sides and yields a sentence, thus being the category of *transitive verb.*

Categorial grammar, that started from works of Ajdukiewicz [4] and Bar-Hillel [5], aspires to practice linguistics to the standards of mathematical logic; for example, Lambek [23] proves cut-elimination, that yields the subformula property, decidability, the finite reading property, and the focalisation property. In a remarkable series of works Mati Pentus has proved the main metatheoretical results for Lambek calculus: equivalence to context free grammars [31]; completeness w.r.t. language models [32][33]; NP-completeness [34]; a polynomial-time algorithm for checking provability of formulae of bounded order in $\mathbf{L}^*$ [35]. The Lambek calculus with only one division operation (and without product) is decidable in polynomial time (Savateev [36]).

The Lambek calculus with brackets **Lb** (Morrill 1992 [27]; Moortgat 1995 [25]) is a logic of bracketed strings which is a conservative extension of Lambek calculus with bracket modalities the rules for which are conditioned on metasyntactic brackets. In this paper we consider a variant of **Lb** that allows empty antecedents, denoted by $\mathbf{Lb}^*$.

The syntax of $\mathbf{Lb}^*$ is more involved than the syntax of the original Lambek calculus. In **L**, the antecedent (left-hand side) of a sequent is just a linearly ordered sequence of formulae. In $\mathbf{Lb}^*$, it is a structure called *configuration,* or *meta-formula.* Meta-formulae are built from formulae, or *types,* as they are called in categorial grammar, using two metasyntactic constructors: comma and brackets. The succedent (right-hand side) of a sequent is one type. Types, in turn, are built from variables, or *primitive types,* $p_1, p_2, \ldots$, using the three binary connectives of Lambek, $\setminus$, $/$, and $\cdot$, and two unary ones, $\langle\rangle$ and $[]^{-1}$, that operate brackets. Axioms of $\mathbf{Lb}^*$ are $p_i \to p_i$, and the rules are as follows:

$$\frac{\Pi \to A \quad \Delta(B) \to C}{\Delta(\Pi, A \setminus B) \to C}\ (\setminus \to) \qquad \frac{A, \Pi \to B}{\Pi \to A \setminus B}\ (\to \setminus) \qquad \frac{\Gamma(A, B) \to C}{\Gamma(A \cdot B) \to C}\ (\cdot \to)$$

$$\frac{\Pi \to A \quad \Delta(B) \to C}{\Delta(B / A, \Pi) \to C}\ (/ \to) \qquad \frac{\Pi, A \to B}{\Pi \to B / A}\ (\to /) \qquad \frac{\Gamma \to A \quad \Delta \to B}{\Gamma, \Delta \to A \cdot B}\ (\to \cdot)$$

$$\frac{\Delta([A]) \to C}{\Delta(\langle\rangle A) \to C}\ (\langle\rangle \to) \qquad \frac{\Pi \to A}{[\Pi] \to \langle\rangle A}\ (\to \langle\rangle) \qquad \frac{\Delta(A) \to C}{\Delta([[]^{-1} A]) \to C}\ ([]^{-1} \to) \qquad \frac{[\Pi] \to A}{\Pi \to []^{-1} A}\ (\to []^{-1})$$

Cut-elimination is proved in Moortgat [25]. The Lambek calculus with brackets permits the characterisation of syntactic domains in addition to word order. By way of linguistic example, consider how a relative pronoun type assignment $(CN \setminus CN) / (S / N)$ (here CN is *one* primitive type, corresponding to *common noun:* e.g., "*book*", as opposed to noun phrase "*the book*") allows unbounded relativisation by associative assembly of the body of relative clauses:

(a) *man who Mary likes*
(b) *man who John knows Mary likes*
(c) *man who Mary knows John knows Mary likes* ...

Thus, (b) is generated because the following is a theorem in the pure Lambek calculus:

$$CN, (CN \setminus CN) / (S / N), N, (N \setminus S) / S, N, (N \setminus S) / N \to CN$$

Consider also, however, the following example: **book which John laughed without reading*, where * indicates that this example is not grammatical. In the original Lambek calculus this ungrammatical example is generated, but in Lambek calculus with brackets its ungrammaticality can be characterised by assigning the adverbial preposition a type $[]^{-1}((N \setminus S) \setminus (N \setminus S)) / (N \setminus S)$ blocking this phrase because the following is not a theorem in Lambek calculus with brackets:

$$CN, (CN \setminus CN) / (S / N), N, N \setminus S, [[]^{-1}((N \setminus S) \setminus (N \setminus S)) / (N \setminus S), (N \setminus S) / N] \to CN$$

where the $[]^{-1}$ engenders brackets which block the associative assembly of the body of the relative clause. Another example of islands is provided by the "and" ("or") construction: **girl whom John loves Mary and Peter loves.*

Jäger [16] claims to prove the context free equivalence of **Lb** grammar on the basis of a translation from **Lb** to **L** due to Michael Moortgat's student Koen Versmissen [37]. However, contrary to Versmissen the translation is not an embedding translation (Fadda and Morrill [11], p. 124). We present the counter-example in the end of Section 3. Consequently the result of Jäger is in doubt: the context free equivalence theorem might be correct, but the proof of Jäger, resting on the Versmissen translation, is not correct.

Pentus [35], following on Aarts [1], presents an algorithm for provability in $\mathbf{L}^*$ based on tabularisation (memoisation) of proof nets. This algorithm runs in polynomial time, if the order of the sequent is bounded. An algorithm of the same kind was also developed by Fowler [12][13] for the Lambek calculus without product. For the unbounded case, the derivability problem for $\mathbf{L}^*$ is in the NP class and is NP-complete [34]. Also, non-associative Lambek calculus [24] can be embedded into the Lambek calculus with brackets (Kurtonina [19]). De Groote [10], following on Aarts and Trautwein [2], showed polynomial-time decidability of the non-associative Lambek calculus. The Lambek calculus with bracket modalities, $\mathbf{Lb}^*$, includes as subsystems both non-associative and associative Lambek calculi.

In this paper we provide a Pentus-style algorithm for $\mathbf{Lb}^*$ provability using (1) the proof nets for Lambek calculus with brackets of Fadda and Morrill [11] which are based on a correction of the Versmissen translation, and (2) an automata-theoretic argument. Again, for the unbounded case, $\mathbf{Lb}^*$ is NP-hard (since it contains $\mathbf{L}^*$ as a conservative fragment), and also belongs to the NP class, since the size of a cut-free derivation in $\mathbf{Lb}^*$ is linearly bounded by the size of the goal sequent.

The rest of this paper is organised as follows. In Section 2 we define complexity parameters and formulate the main result. Section 3 contains the formulation and proof of a graph-theoretic provability criterion for $\mathbf{Lb}^*$, known as proof nets. In Section 4 we introduce some more convenient complexity parameters and show their polynomial equivalence to the old ones. Section 5 is the central one, containing the description of our algorithm. In order to make this paper self-contained, in Section 6 we give a detailed explanation of Pentus' construction [35], since it is crucial for our algorithm to work. Finally, in Section 7 we discuss directions of future research in this field.

2 The Main Result

For a sequent $\Gamma \to C$ we consider the following three *complexity parameters*. The first one is the *size* of the sequent, $\|\Gamma \to C\|$, counted as the total number of variables and logical symbols in it, including brackets.

▶ **Definition 1.** The size of a formula, meta-formula, or sequent in $\mathbf{Lb}^*$ is defined recursively as follows: $\|p_i\| = 0$; $\|A \cdot B\| = \|A \setminus B\| = \|B / A\| = \|A\| + \|B\| + 1$; $\|\langle\rangle A\| = \|[]^{-1} A\| = \|A\| + 1$; $\|\Lambda\| = 0$; $\|\Gamma, \Delta\| = \|\Gamma\| + \|\Delta\|$; $\|[\Gamma]\| = \|\Gamma\| + 2$; $\|\Gamma \to C\| = \|\Gamma\| + \|C\|$.

The second parameter is the *order*.

▶ **Definition 2.** For any formula A let $\mathrm{prod}(A)$ be 1 if A is of the form $A_1 \cdot A_2$ or $\langle\rangle A_1$, and 0 if not. The order of a formula, meta-formula, or sequent in $\mathbf{Lb}^*$ is defined recursively: $\mathrm{ord}(p_i) = 0$; $\mathrm{ord}(A \cdot B) = \max\{\mathrm{ord}(A), \mathrm{ord}(B)\}$; $\mathrm{ord}(A \setminus B) = \mathrm{ord}(B \,/\, A) = \max\{\mathrm{ord}(A) + 1, \mathrm{ord}(B) + \mathrm{prod}(B)\}$; $\mathrm{ord}(\langle\rangle A) = \mathrm{ord}(A)$; $\mathrm{ord}([]^{-1}A) = \max\{\mathrm{ord}(A) + \mathrm{prod}(A), 1\}$; $\mathrm{ord}(\Lambda) = 0$; $\mathrm{ord}(\Gamma, \Delta) = \max\{\mathrm{ord}(\Gamma), \mathrm{ord}(\Delta)\}$; $\mathrm{ord}([\Gamma]) = \mathrm{ord}(\Gamma)$; $\mathrm{ord}(\Gamma \to C) = \max\{\mathrm{ord}(\Gamma) + 1, \mathrm{ord}(C) + \mathrm{prod}(C)\}$.

For sequents without $\cdot$ and $[]^{-1}$, this definition is quite intuitive: the order is the nesting depth of implications ($\setminus$, $/$, and finally $\to$) and $\langle\rangle$ modalities. With $\cdot$, we also count *alternations* between divisions and multiplications: for example, in $p_1 \setminus (p_2 \cdot (p_3 \setminus (p_4 \cdot \ldots p_k) \ldots))$ implications are not nested, but the order grows linearly. On the other hand, the order is always bounded by a simpler complexity parameter, the maximal height of the syntactic tree. Also, linguistic applications make use of syntactic types of small, constantly bounded order.

The third parameter is the *bracket nesting depth*.

▶ **Definition 3.** The bracket nesting depth of a formula, meta-formula, or a sequent in $\mathbf{Lb}^*$ is defined recursively as follows: $\mathrm{b}(p_i) = 0$; $\mathrm{b}(A \,/\, B) = \mathrm{b}(B \setminus A) = \mathrm{b}(A \cdot B) = \max\{\mathrm{b}(A), \mathrm{b}(B)\}$; $\mathrm{b}(\langle\rangle A) = \mathrm{b}([]^{-1}A) = \mathrm{b}(A) + 1$; $\mathrm{b}(\Lambda) = 0$; $\mathrm{b}(\Gamma, \Delta) = \max\{\mathrm{b}(\Gamma), \mathrm{b}(\Delta)\}$; $\mathrm{b}([\Gamma]) = \mathrm{b}(\Gamma) + 1$; $\mathrm{b}(\Gamma \to C) = \max\{\mathrm{b}(\Gamma), \mathrm{b}(C)\}$.

By $\mathrm{poly}(x_1, x_2, \ldots)$ we denote a value that is bounded by a polynomial of x_1, x_2, ...

▶ **Theorem 4.** *There exists an algorithm that decides whether a sequent $\Gamma \to C$ is derivable in $\mathbf{Lb}^*$ in $\mathrm{poly}(N, 2^R, N^B)$ time, where $N = \|\Gamma \to C\|$, $R = \mathrm{ord}(\Gamma \to C)$, $B = \mathrm{b}(\Gamma \to C)$.*

If the depth parameters, R and B, are fixed, the working time of the algorithm is polynomial w.r.t. N. However, the dependence on the depth parameters is exponential.

3 Proof Nets

In this section we formulate and prove a graph-theoretic criterion for derivability in $\mathbf{Lb}^*$. A sequent is derivable if and only if there exists a *proof net*, that is, a graph satisfying certain *correctness conditions*.

For each variable p_i we introduce two *literals*, p_i and $\bar{p}_i$, and also four literals, $[$, $]$, $\bar{[}$, and $\bar{]}$ for brackets. Next we define two translations (positive, A^+, and negative, A^-) of $\mathbf{Lb}^*$-formulae into expressions built from literals using two connectives, $\parr$ and $\otimes$.

▶ **Definition 5.**

$$
\begin{aligned}
p_i^+ &= p_i, & p_i^- &= \bar{p}_i,\\
(A \cdot B)^+ &= A^+ \otimes B^+, & (A \cdot B)^- &= B^- \parr A^-,\\
(A \setminus B)^+ &= A^- \parr B^+, & (A \setminus B)^- &= B^- \otimes A^+,\\
(B \,/\, A)^+ &= B^+ \parr A^-, & (B \,/\, A)^- &= A^+ \otimes B^-,\\
(\langle\rangle A)^+ &=] \otimes A^+ \otimes [, & (\langle\rangle A)^- &= \bar{[} \parr A^- \parr \bar{]},\\
([]^{-1}A)^+ &= \bar{]} \parr A^+ \parr \bar{[}; & ([]^{-1}A)^- &= [\otimes A^- \otimes].
\end{aligned}
$$

For meta-formulae, we need only the negative translation. In this translation we use an extra connective, $\diamond$, which serves as a metasyntactic version of $\parr$ (just as the comma is a metasyntactic product in the sequent calculus for $\mathbf{Lb}^*$).

▶ **Definition 6.** $(\Gamma, \Delta)^- = \Delta^- \diamond \Gamma^-$; $\quad [\Gamma]^- = \bar{[} \diamond \Gamma^- \diamond \bar{]}$.

Finally, a sequent $\Gamma \to C$ is translated as $\diamond\Gamma^- \diamond C^+$ (or as $\diamond C^+$ if Γ is empty).

Essentially, this is an extension of Pentus' translation of $\mathbf{L}^*$ into cyclic multiplicative linear logic (**CMLL**) [33][35]. In this paper, for the sake of simpicity, we don't introduce an intermediate calculus that extends **CMLL** with brackets, and we formulate the proof net criterion directly for $\mathbf{Lb}^*$.

Denote the set of all literal and connective *occurrences* in this translation by $\Omega_{\Gamma\to C}$. These occurrences are linearly ordered; connectives and literals alternate. The total number of occurrences is $2n$. Denote the literal occurrences (in their order) by $\ell_1, \ldots, \ell_n$ and the connective occurrences by $c_1, \ldots, c_n$.

▶ **Definition 7.** The *dominance* relation on the occurrences of $\parr$ and $\otimes$, denoted by $\prec$, is defined as follows: for a subexpression of the form $A \parr B$ or $A \otimes B$ if the occurrence of the central connective is c_i, then for any c_j inside A or B we declare $c_j \prec c_i$.

We assume that $\parr$'s that come from translations of bracket modalities associate to the left and $\otimes$'s associate to the right (this choice is arbitrary). Thus, in a pair of such $\parr$'s the right one dominates the left one in the syntactic tree, and the left one dominates the principal connective of A (if A has one, *i.e.,* it is not a literal); symmetrically for $\otimes$.

The other two relations are the *sisterhood* relations on bracket literals and connectives, $\mathcal{S}_b$ and $\mathcal{S}_c$ respectively. Both relations are symmetric.

▶ **Definition 8.** The bracket sisterhood relation, $\mathcal{S}_b$, connects pairs of occurrences of [and] or $\bar{[}$ and $\bar{]}$ that come from the same $\langle\rangle A$, $[]^{-1}A$, or $[\Gamma]$. The connective sisterhood relation, $\mathcal{S}_c$, connects pairs of occurrences of $\otimes$, $\parr$, or $\diamond$ that come from the same $\langle\rangle A$, $[]^{-1}A$, or $[\Gamma]$. Occurrences connected by one of the sisterhood relations will be called *sister* occurrences.

▶ **Definition 9.** A *proof structure* $\mathcal{E}$ is a symmetric relation on the set of literal occurrences $(\{\ell_1, \ldots, \ell_n\})$ such that each occurrence is connected by $\mathcal{E}$ to exactly one occurrence, and each occurrence of a literal q, where q is a variable, [, or], is connected to an occurrence of $\bar{q}$.

▶ **Definition 10.** A proof structure $\mathcal{E}$ is *planar,* iff its edges can be drawn in a semiplane without intersection while the literal occurrences are located on the border of this semiplane in their order $(\ell_1, \ldots, \ell_n)$.

Edges of a planar proof structure divide the upper semiplane into *regions.* The number of regions is $\frac{n}{2} + 1$ (the outermost, infinite region also counts).

▶ **Definition 11.** A planar proof structure $\mathcal{E}$ is a *proof net,* iff it satisfies two conditions.

1. On the border of each region there should be exactly one occurrence of $\parr$ or $\diamond$.
2. Define an oriented graph $\mathcal{A}$ that connects each occurrence of $\otimes$ to the unique occurrence of $\parr$ or $\diamond$ located in the same region. The graph $\mathcal{A} \cup \prec$ should be acyclic.

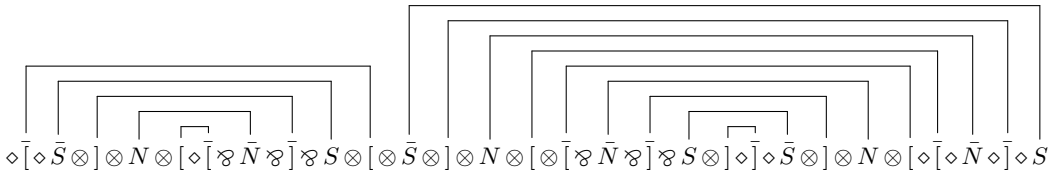
By definition, edges of $\mathcal{E}$ and $\mathcal{A}$ in a proof net do not intersect, in other words, the graph $\mathcal{E} \cup \mathcal{A}$ is also planar. We can also consider proof structures and proof nets on expressions that are not translations of $\mathbf{Lb}^*$ sequents. For example, proof nets allow *cyclic permutations:* a proof net for $\diamond\gamma_1 \diamond\gamma_2$ can be transformed into a proof net for $\diamond\gamma_2 \diamond\gamma_1$ (the $\prec$ relation in γ_1 and γ_2 is preserved).

▶ **Definition 12.** A proof structure $\mathcal{E}$ *respects sisterhood,* iff the following condition holds: if $\langle \ell_i, \ell_{i'} \rangle \in \mathcal{E}$, $\langle \ell_i, \ell_j \rangle \in \mathcal{S}_b$, and $\langle \ell_{i'}, \ell_{j'} \rangle \in \mathcal{S}_b$, then $\langle \ell_j, \ell_{j'} \rangle \in \mathcal{E}$ (i.e., sister brackets are connected to sister brackets).

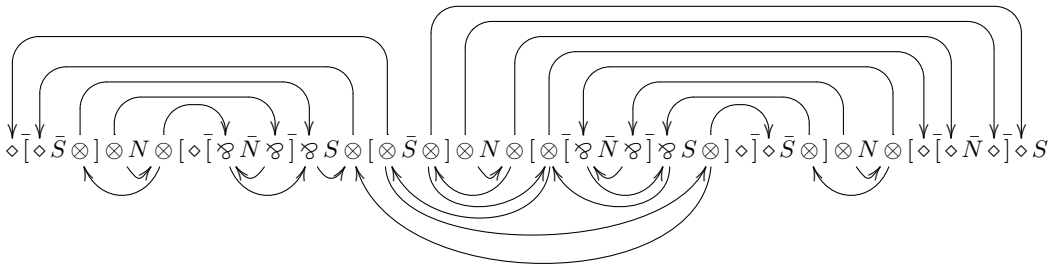
Before continuing, we consider an **example** of a proof net for a sequent with linguistic meaning. According to [28], the sentence *"Mary danced before singing"* gets the following bracketing: *"*[*Mary*] *danced* [*before singing*]*"* and the following type assignment:

$$[N], \langle\rangle N \setminus S, [[]^{-1}((\langle\rangle N \setminus S) \setminus (\langle\rangle N \setminus S)) / (\langle\rangle N \setminus S), \langle\rangle N \setminus S] \to S$$

This sequent is derivable in $\mathbf{Lb}^*$, and we justify it by presenting a proof net. This is achieved by translating the sequent into a string of literals according to Definition 5 and drawing an appropriate $\mathcal{E}$ graph that satisfies all the conditions for being a proof net (Definition 11):

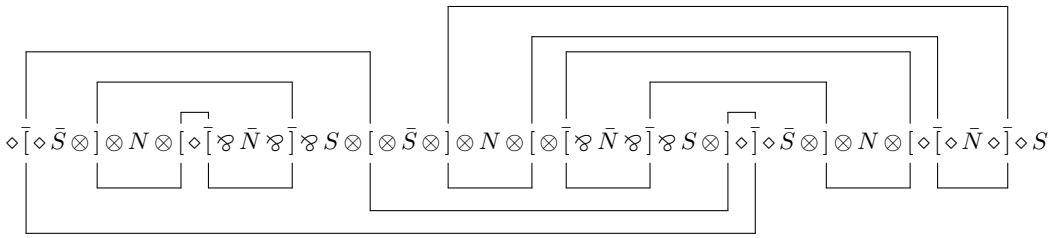


Informally speaking, two literals are connected by $\mathcal{E}$ if they come from the same axiom or bracket rule instance. The next figure depicts the $\mathcal{A}$ relation obtained from $\mathcal{E}$ by Definition 11 and the dominance relation $\prec$ (Definition 7), showing that $\mathcal{A} \cup \prec$ is acyclic:



Here $\mathcal{A}$ and $\prec$ are drawn above and below the string respectively.

Finally, as one can see from the figure below, the proof net from our example respects sisterhood:



Here $\mathcal{E}$ (we keep only edges connected to brackets) and $\mathcal{S}_b$ are drawn above and below the string respectively. Graphically the sisterhood condition means that these edges form 4-cycles. In the end of this section we show that this sisterhood condition is essential: if it fails, the sequent could be not derivable.

Now we are going to prove that a sequent is derivable in $\mathbf{Lb}^*$ if and only if there exists a proof net that respects sisterhood. First we establish the following technical lemma.

▶ **Lemma 13.** *For an expression of the form $\diamond A_1^- \diamond \ldots \diamond A_m^-$ (which is not a translation of an $\mathbf{Lb}^*$ sequent, since there is no B^+ at the end) there couldn't exist a proof net.*

Proof. Following Pentus [33], for any string γ of literals, $\parr$'s, $\otimes$'s, and $\diamond$'s we define $\natural(\gamma)$ as the number of negative literals (i.e., of the form $\bar{q}$, where q is a variable or a bracket) minus

the number of $\parr$'s and $\diamond$'s. Then we establish the following: (1) $\natural(A^+) = 0$ and $\natural(A^-) = 1$ for any formula A; (2) if there exists a proof net for γ, then $\natural(\gamma) = -1$. The first statement is proved by joint induction on A. The second one follows from the fact that the number of regions is greater than the number of $\mathcal{E}$ links exactly by one; links are in one-to-one correspondence with negative literal occurrences and each region holds a unique occurrence of $\parr$ or $\diamond$. Since $\natural(\diamond A_1^- \diamond \ldots \diamond A_m^-) = m - m = 0 \neq -1$, there is no proof net. ◀

▶ **Theorem 14.** *The sequent $\Gamma \to C$ is derivable in* **Lb*** *if and only if there exists a proof net $\mathcal{E}$ over $\Omega_{\Gamma\to C}$ that respects sisterhood.*

Proof. The direction from **Lb***-derivation to proof net is routine: we construct the proof net by induction, maintaining the correctness criterion. For the other direction, we proceed by induction on the number of $\parr$ and $\otimes$ occurrences.

If there are no occurrences of $\parr$ or $\otimes$, then the total number of $\diamond$ occurrences is, on the one hand, equal to n; on the other hand, it is equal to the number of regions, $\frac{n}{2} + 1$. Therefore, $n = 2$, and the only possible proof net is $\diamond\, \bar{p}\, \diamond\, p$ that corresponds to the $p \to p$ axiom.

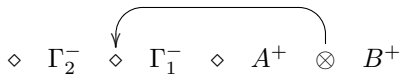
Otherwise consider the set of all occurrences of $\parr$ and $\otimes$ with the relation $\mathcal{A} \cup \prec$. Since this relation is acyclic (and the set is not empty), there exists a maximal element, c_i.

Case 1.1: c_i is a $\parr$ occurrence that came from $(A \cdot B)^- = B^- \parr A^-$. Replacing this $\parr$ by $\diamond$ corresponds to applying $(\cdot \to)$.

Case 1.2: c_i is a $\parr$ occurrence that came from $(A \setminus B)^+ = A^- \parr B^+$. Replacing this $\parr$ by $\diamond$ changes $\diamond\Gamma^- \diamond A^- \parr B^+$ to $\diamond\Gamma^- \diamond A^- \diamond B^+$, which corresponds to applying $(\to \setminus)$. (In the negative translation, formulae in the left-hand side appear in the inverse order.)

Case 1.3: c_i is a $\parr$ occurrence that came from $(B \,/\, A)^+ = B^+ \parr A^-$. Again, replace $\parr$ with $\diamond$ and cyclically transform the net, yielding $\diamond A^- \diamond\Gamma^- \diamond B^+$. Then apply $(\to /)$.

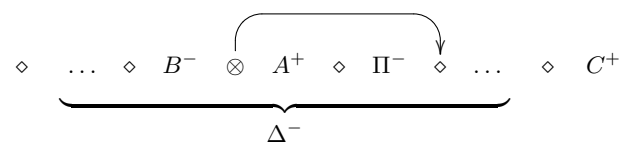
Case 2.1: c_i is a $\otimes$ occurrence that came from $(A \cdot B)^+ = A^+ \otimes B^+$. Then $\mathcal{A}(c_i)$ is a $\diamond$ occurrence, and the $\mathcal{A}$ link splits the proof net for $\Gamma \to A \cdot B$ into two separate proof nets for $\Gamma_1 \to A$ and $\Gamma_2 \to B$ (Γ_1 and/or Γ_2 could be empty, then two or three $\diamond$'s shrink into one):



Note that here the fragments before the $\diamond$ occurrence $\mathcal{A}(c_i)$ and between $\mathcal{A}(c_i)$ and A are negative translations of whole metaformulae (Γ_1 and Γ_2), not just substrings with possibly disbalanced brackets. Indeed, suppose that a pair of sister brackets, $\bar{[}$ and $\bar{]}$, is split between these two fragments. Then, since $\mathcal{E}$ links cannot intersect $\mathcal{A}$, the corresponding pair of [and], connected to the original pair by $\mathcal{E}$, will also be split and therefore belong to translations of different formulae. However, they also form a sister pair (our proof net respects sisterhood), and therefore should belong to one formula (by definition of the translation). Contradiction.

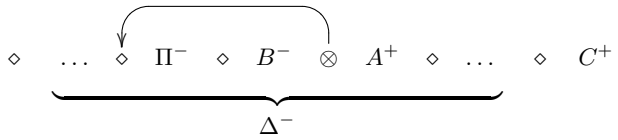
Since for the sequents $\Gamma_1 \to A$ and $\Gamma_2 \to B$ the induction parameter is smaller, they are derivable, and therefore $\Gamma_1, \Gamma_2 \to A \cdot B$ is derivable by application of the $(\to \cdot)$ rule.

Case 2.2: c_i is a $\otimes$ occurrence from $(A \setminus B)^- = B^- \otimes A^+$. Again, the proof net gets split:



(As in the previous case, no pair of sister brackets could be split by $\mathcal{A}$ here, and Π^- is a translation of a whole metaformula.) The outer fragment provides a proof net for $\Delta\langle B\rangle \to C$; applying a cyclic permutation to the inner fragment yields a proof net for $\Pi \to A$. The goal sequent, $\Delta\langle \Pi, A \setminus B\rangle \to C$, is obtained by applying the $(/ \to)$ rule.

The other situation,



is impossible by Lemma 13, applied to the inner net.

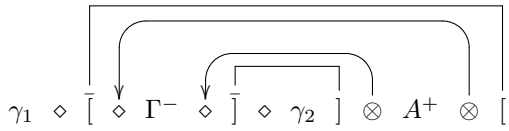
Case 2.3: c_i is a $\otimes$ occurrence that came from $(B \,/\, A)^- = A^+ \otimes B^-$. Symmetric.

Case 3.1: c_i is a $\parr$ occurrence that came from $(\square^{-1}A)^+ = \overline{]} \parr A^+ \parr \overline{[}$. Then we replace two $\parr$'s by $\diamond$'s and cyclically relocate the rightmost $\overline{[}$ with its $\mathcal{E}$ link, obtaining $\diamond \overline{[} \diamond \Gamma^- \diamond \overline{]} \diamond A^+$ from $\diamond \Gamma^- \diamond \overline{]} \parr A^+ \parr \overline{[}$. This corresponds to an application of $(\to \square^{-1})$.

Case 3.2: c_i is a $\parr$ occurrence that came from $(\langle\rangle A)^- = \overline{[} \parr A^- \parr \overline{]}$. By replacing $\parr$'s with $\diamond$'s, we change $\langle\rangle A$ into $[A]$. This corresponds to an application of $(\langle\rangle \to)$.

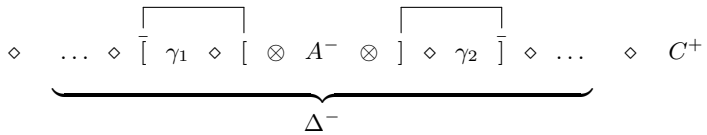
Case 4.1: c_i is a $\otimes$ occurrence that came from $(\langle\rangle A)^+ =] \otimes A^+ \otimes [$. Consider the $\mathcal{E}$ links that go from these] and [. Since $(\langle\rangle A)^+$ is the rightmost formula, they both either go to the left or into A^+. The second situation is impossible, because then $\mathcal{A}(c_i)$ should also be a $\parr$ occurrence in A^+, that violates the maximality of c_i (and also the acyclicity condition).

In the first situation, the picture is as follows:



(Due to maximality of c_i, $\mathcal{A}(c_i)$ and its sister are $\diamond$'s, not $\parr$'s.) Clearly, γ_1 and γ_2 are empty: otherwise we have two $\diamond$'s in one region. Then we can remove brackets and transform this proof net into a proof net for $\diamond\Gamma^- \diamond A^+$, i.e., $\Gamma \to A$. Applying $(\to \langle\rangle)$ yields $[\Gamma] \to \langle\rangle A$.

Case 4.2: c_i is a $\otimes$ occurrence that came from $(\square^{-1}A)^- = [\otimes A^- \otimes]$. As in the previous case, consider the $\mathcal{E}$ links going from these bracket occurrences. The good situation is when they go to different sides:



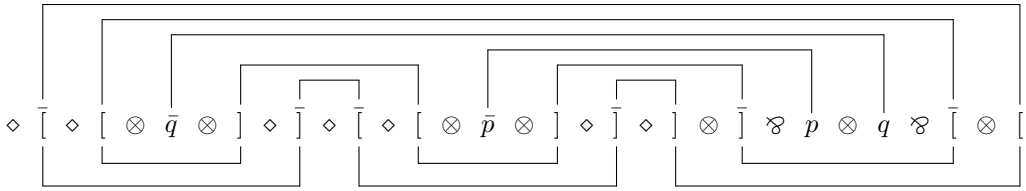
(The connectives surrounding $[\otimes A^- \otimes]$ are $\diamond$'s due to the maximality of c_i.) Again, γ_1 and γ_2 should be empty (the connective after $\overline{[}$ or before $\overline{]}$ here cannot be a $\otimes$, and we get more than one $\parr$ or $\diamond$ in a region), and removing the bracket corresponds to applying $(\square^{-1} \to)$: replace $[\square^{-1}A]$ with A in the context Δ.

Potentially, the $\mathcal{E}$ links from the brackets could also go to one side, but then they end at a pair of sister brackets $\overline{]}$ and $\overline{[}$ (in this order), which can occur only in $\overline{]} \parr B \parr \overline{[}$. Then one of these $\parr$'s is $\mathcal{A}(c_i)$, which contradicts the maximality of c_i. ◀

The idea of proof nets as a representation of derivation in a parallel way comes from Girard's original paper on linear logic [15]. For the non-commutative case, including the Lambek calculus, proof nets were studied by many researchers including Abrusci [3], de Groote [9], Nagayama and Okada [29], Penn [30], Pentus [33], Yetter [38], and others. In our definition

of proof nets for $\mathbf{Lb}^*$ we follow Fadda and Morrill [11], but with the correctness (acyclicity) conditions of Pentus [33][35] rather than Danos and Regnier [8].

The idea of handling brackets similarly to variables is due to Versmissen [37]. If we take a sequent that is derivable in $\mathbf{Lb}^*$, replace brackets with fresh variables, say, r and s, and respectively substitute $r \cdot A \cdot s$ for $\langle\rangle A$ and $r \setminus A / s$ for $[]^{-1}A$, we obtain a sequent that is derivable in $\mathbf{L}^*$ (this follows from our proof net criterion and can also be shown directly). Versmissen, however, claims that the converse is also true. This would make our Theorem 4 a trivial corollary of Pentus' result [35], but Fadda and Morrill [11] present a counter-example to Versmissen's claim. Namely, the sequent $[]^{-1}p], []^{-1}q] \to \langle\rangle[]^{-1}(p \cdot q)$ is not derivable in $\mathbf{Lb}^*$, but its translation, $r, r \setminus p / s, s, r, r \setminus q / s, s \to r \cdot (r \setminus (p \cdot q) / s) \cdot s$, is derivable in $\mathbf{L}^*$. This example shows the importance of the sisterhood condition in Theorem 14: the only possible proof net for this sequent, shown below, doesn't respect sisterhood.



4 Complexity Parameters for Proof Nets

In this section we introduce new complexity parameters that operate with $\Omega_{\Gamma\to C}$ rather than with the original sequent $\Gamma \to C$, and therefore are more handy for complexity estimations. We show that a value is polynomial in terms of the old parameters if it is polynomial in terms of the new ones.

The first parameter, denoted by n, is the number of literals in $\Omega_{\Gamma\to C}$. It is connected to the size of the original sequent by the following inequation: $n \le 2\|\Gamma \to C\|$. (We have to multiply by 2, since a modality, $\langle\rangle$ or $[]^{-1}$, being counted as one symbol in $\Gamma \to C$, introduces two literals.)

The second parameter, denoted by $d = d(\Omega_{\Gamma\to C})$, is the *connective alternation depth,* and informally it is the maximal number of alternations between $⅋$ and $\otimes$ on the $\prec$-path from any literal to the root of the parse tree. Formally it is defined by recursion.

▶ **Definition 15.** For an expression γ constructed from literals using $⅋$, $\otimes$, and $\diamond$, let $\mathrm{prod}(\gamma)$ be 1 if γ is of the form $\gamma_1 \otimes \gamma_2$, and 0 otherwise. Define $d(\gamma)$ by recursion: $d(q) = d(\bar{q}) = 0$ for any literal q; $d(\gamma_1 ⅋ \gamma_2) = d(\gamma_1 \diamond \gamma_2) = \max\{d(\gamma_1) + \mathrm{prod}(\gamma_1), d(\gamma_2) + \mathrm{prod}(\gamma_2)\}$; $d(\gamma_1 \otimes \gamma_2) = \max\{d(\gamma_1), d(\gamma_2)\}$.

The d parameter is connected to the order of the original sequent:

▶ **Lemma 16.** *For any sequent $\Gamma \to C$, the following holds: $d(\Omega_{\Gamma\to C}) \le \mathrm{ord}(\Gamma \to C)$.*

Proof. We prove the following two inequations for any formula A by simultaneous induction on the construction of A: $d(A^+) \le \mathrm{ord}(A)$ and $d(A^-) + \mathrm{prod}(A^-) \le \mathrm{ord}(A)$. The induction is straightforward, the only interesting cases are the negative ones for $A_2 \setminus A_1$, A_1 / A_2, and $([]^{-1}A_1)^-$. In those three cases, we need to branch further into two subcases: whether A_1 is a variable or a complex type. Finally, $d(\Omega_{\Gamma\to C}) = d(\diamond A_1^- \diamond \ldots \diamond A_k^- \diamond C^+) = \max\{d(A_1^-) + \mathrm{prod}(A_1^-), \ldots, d(A_k^-) + \mathrm{prod}(A_k^-), d(C^+) + \mathrm{prod}(C^+)\} \le \max\{\mathrm{ord}(\Gamma), \mathrm{ord}(C) + \mathrm{prod}(C)\} \le \max\{\mathrm{ord}(\Gamma) + 1, \mathrm{ord}(C) + \mathrm{prod}(C)\} = \mathrm{ord}(\Gamma \to C)$. ◀

The third parameter is b, the maximal nesting depth of pairs of sister brackets. Clearly, $b = \mathrm{b}(\Gamma \to C)$.

In view of the inequations established in this section, if a value is $\mathrm{poly}(n, 2^d, n^b)$, it is also $\mathrm{poly}(\|\Gamma \to C\|, 2^{\mathrm{ord}(\Gamma\to C)}, \|\Gamma \to C\|^{\mathrm{b}(\Gamma\to C)})$, and for our algorithm we'll establish complexity bounds in terms of n, d, and b.

5 The Algorithm

Our goal is to obtain an efficient algorithm that searches for proof nets that respect sisterhood. We are going to split this task: first find all possible proof nets satisfying Pentus' correctness conditions, and then distill out those which respect sisterhood. One cannot, however, simply yield all the proof nets. The reason is that there exist derivable sequents, even without brackets and of order 2, that have exponentially many proof nets, for example, $p \,/\, p, \ldots, p \,/\, p, p, p \setminus p, \ldots, p \setminus p \to p$. Therefore, instead of generating all the proof nets for a given sequent, Pentus, as a side-effect of his provability verification algorithm, produces a context free grammar that generates a set of words encoding all these proof nets. We filter this set by intersecting it with the set of codes of all proof structures that respect sisterhood. For the latter, we build a finite automaton of polynomial size.

Note that this context free grammar construction is *different* from the translation of Lambek categorial grammars into context free grammars (Pentus [31]). The grammar from Pentus' algorithm that we consider here generates all proof nets for a *fixed* sequent, while in [31] a context free grammar is generated for *all* words that have corresponding derivable Lambek sequents. The latter (global) grammar is of exponential size (though for the case of only one division there also exists a polynomial construction [21]), while the former (local) one is polynomial. For the bracket extension, we present a construction of the local grammar. The context-freeness for the global case is claimed by Jäger [16], but his proof uses the incorrect lemma by Versmissen (see above).

Following Pentus [35], for a given sequent $\Gamma \to C$ we encode proof structures as words of length n over alphabet $\{e_1, \ldots, e_n\}$.

▶ **Definition 17.** The code $c(\mathcal{E})$ of proof structure $\mathcal{E}$ is constructed as follows: if ℓ_i and ℓ_j are connected by $\mathcal{E}$, then the i-th letter of $c(\mathcal{E})$ is e_j and the j-th letter is e_i.

The code of a proof structure is always an involutive permutation of $e_1, \ldots, e_n$.

We are going to define two languages, P_1 and P_2, with the following properties:

1. $P_1 = \{c(\mathcal{E}) \mid \mathcal{E} \text{ is a proof net}\}$;
2. $c(\mathcal{E}) \in P_2$ iff $\mathcal{E}$ respects sisterhood.

Note that in the condition for P_2 we say nothing about words that are not of the form $c(\mathcal{E})$. Some of these words could also belong to P_2. Nevertheless, $w \in P_1 \cap P_2$ iff $w = c(\mathcal{E})$ for some pairing $\mathcal{E}$ that is a proof net and respects sisterhood. Therefore, the sequent is derivable in $\mathbf{Lb}^*$ iff $P_1 \cap P_2 \neq \varnothing$.

Now the algorithm that checks derivability in $\mathbf{Lb}^*$ works as follows: it constructs a context free grammar for $P_1 \cap P_2$ and checks whether the language generated by this grammar is non-empty. Notice that the existence of such a grammar is trivial, since the language is finite. However, it could be of exponential size, and we're going to construct a grammar of size $\mathrm{poly}(n, 2^d, n^b)$, and do it in polynomial time.

For P_1, we use the construction from [35]. As the complexity measure (size) of a context free grammar, $|\mathfrak{G}_1|$, we use the sum of the length of its rules.

▶ **Theorem 18** (M. Pentus 2010). *There exists a context free grammar* $\mathfrak{G}_1$ *of size* $\mathrm{poly}(n, 2^d)$ *that generates* P_1. *Moreover, this grammar can be obtained from the original sequent by an algorithm with working time also bounded by* $\mathrm{poly}(n, 2^d)$.

This theorem is stated as a remark in [35]. We give a full proof of it in Section 6.

Next, we construct a finite automaton for a language that satisfies the condition for P_2.

▶ **Lemma 19.** *There exists a deterministic finite automaton with* $\mathrm{poly}(n, n^b)$ *states that generates a language* P_2 *over alphabet* $\{e_1, \ldots, e_n\}$ *such that* $c(\mathcal{E}) \in P_2$ *iff* $\mathcal{E}$ *respects sisterhood. Moreover, this finite automaton can be obtained from the original sequent by an algorithm with working time* $\mathrm{poly}(n, n^b)$.

Proof. First we describe this automaton informally. Its memory is organised as follows: it includes a pointer i to the current letter of the word (a number from 1 to $n+1$) and a stack that can be filled with letters of $\{e_1, \ldots, e_n\}$. In the beginning, $i = 1$ and the stack is empty. At each step (while $i \leq n$), the automaton looks at ℓ_i. If it is not a bracket, the automaton increases the pointer and proceeds to the next letter in the word. If it is a bracket, let its sister bracket be ℓ_j. Denote the i-th (currently being read) letter of the word by $e_{i'}$. If $\ell_{i'}$ is not a bracket, yield "no" (bracket is connected to non-bracket). Otherwise let $\ell_{j'}$ be the sister of $\ell_{i'}$ and consider two cases.

1. $j > i$. Then push $e_{j'}$ on top of the stack, increase the pointer and continue.
2. $j < i$. Then pop the letter from the top of the stack and compare it with $e_{i'}$. If they do not coincide, yield "no". Otherwise increase the pointer and continue.

If $i = n + 1$ and the stack is empty, yield "yes".

Since sister brackets are well-nested, on the i-th step we pop from the stack the symbol that was pushed there on the j-th step (if ℓ_i and ℓ_j are sister brackets and $j < i$). Thus, the symbol popped from the stack contains exactly the information that, if the bracket ℓ_j is connected to $\ell_{j'}$, then the bracket ℓ_i should be connected to the sister bracket $\ell_{i'}$, and we verify the fact that $\mathcal{E}$ satisfies this condition by checking that the i-th letter is actually $e_{i'}$.

Note that here we do not check the fact that the word really encodes some proof structure $\mathcal{E}$, since malformed codes will be ruled out by the intersection with P_1.

If the bracket nesting depth is b, we'll never have more than b symbols on the stack. For each symbol we have n possibilities (e_1, …, e_n). Therefore, the total number of possible states of the stack is $1 + n + n^2 + \ldots + n^b \leq (b+1) \cdot n^b$. The pointer has $(n+1)$ possible values. Thus, the whole number of possible memory states is $(n+1) \cdot (b+1) \cdot n^b + 1$ (the last "+1" is for the "failure" state, in which the automaton stops to yield "no").

Formally, our automaton is a tuple $\mathfrak{A}_2 = \langle Q, \Sigma, \delta, q_0, \{q_F\}\rangle$, where $\Sigma = \{e_1, \ldots, e_n\}$ is the alphabet, $Q = \{1, \ldots, n+1\} \times \Sigma^{\leq b} \cup \{\bot\}$, where $\Sigma^{\leq b}$ is the set of all words over Σ of length not greater than b, is the set of possible states ($\bot$ is the "failure" state), $q_0 = \langle 1, \varepsilon\rangle$ is the initial state, $q_F = \langle n+1, \varepsilon\rangle$ is the final (accepting) state, and $\delta \subset Q \times \Sigma \times Q$ is a set of transitions defined as follows:

$$\begin{aligned}
\delta = {} & \{\langle i, \xi\rangle \xrightarrow{e_{i'}} \langle i+1, \xi\rangle \mid \ell_i \text{ is not a bracket}\} \\
& \cup \{\langle i, \xi\rangle \xrightarrow{e_{i'}} \bot \mid \ell_i \text{ is a bracket and } \ell_{i'} \text{ is not a bracket}\} \\
& \cup \{\langle i, \xi\rangle \xrightarrow{e_{i'}} \langle i+1, \xi e_{j'}\rangle \mid \ell_i \text{ is a bracket, its sister bracket is } \ell_j,\ j > i; \\
& \qquad\qquad \ell_{i'} \text{ is a bracket, its sister bracket is } \ell_{j'}\} \\
& \cup \{\langle i, \xi e_{i'}\rangle \xrightarrow{e_{i'}} \langle i+1, \xi\rangle \mid \ell_i \text{ is a bracket, its sister bracket is } \ell_j,\ j < i\} \\
& \cup \{\langle i, \xi e_{i''}\rangle \xrightarrow{e_{i'}} \bot \mid \ell_i \text{ is a bracket, its sister bracket is } \ell_j,\ j < i, \text{ and } i' \neq i''\}.
\end{aligned}$$

$\mathfrak{A}_2$ is a deterministic finite automaton with not more than $(n+1)\cdot(b+1)\cdot n^b+1 = \mathrm{poly}(n, n^b)$ states, and it generates a language P_2 such that $c(\mathcal{E}) \in P_2$ iff $\mathcal{E}$ respects sisterhood.

In the RAM model, each transition is computed in constant time, and the total number of transitions is not more than $|Q|^2 \cdot n \leq ((n+1)\cdot(b+1)\cdot n^b+1)^2 \cdot n$, which is also $\mathrm{poly}(n, n^b)$. ◀

Now we combine Theorem 18 and Lemma 19 to obtain a context free grammar $\mathfrak{G}$ for $P_1 \cap P_2$ of size $\mathrm{poly}(|\mathfrak{G}_1|, |\mathfrak{A}_2|, |\Sigma|)$, where $|\mathfrak{A}_2|$ is the number of states of $\mathfrak{A}_2$. For this we use the following well-known result:

▶ **Theorem 20.** *If a context free grammar $\mathfrak{G}_1$ defines a language P_1 over an alphabet Σ and a deterministic finite automaton $\mathfrak{A}_2$ defines a language P_2 over the same alphabet, then there exists a context free grammar $\mathfrak{G}$ that defines $P_1 \cap P_2$, the size of this grammar is $\mathrm{poly}(|\mathfrak{G}_1|, |\mathfrak{A}_2|, |\Sigma|)$, and, finally, this grammar can be obtained from $\mathfrak{G}_1$ and $\mathfrak{A}_2$ by an algorithm with working time also $\mathrm{poly}(|\mathfrak{G}_1|, |\mathfrak{A}_2|, |\Sigma|)$.*

For this theorem we use the construction from [14, Theorem 3.2.1] that works directly with the context free formalism. This makes the complexity estimation straightforward. Since $|\mathfrak{G}_1| = \mathrm{poly}(n, 2^d)$, $|\mathfrak{A}_2| = \mathrm{poly}(n, n^b)$, and $|\Sigma| = n$, $|\mathfrak{G}_2|$ is $\mathrm{poly}(n, 2^d, n^b)$. Finally, checking derivability of the sequent is equivalent to testing the language $P_1 \cap P_2$ for non-emptiness, which is done using the following theorem [14, Lemma 1.4.3a and Theorem 4.1.2a]:

▶ **Theorem 21.** *There exists an algorithm that checks whether the language generated by a context free grammar $\mathfrak{G}$ is non-empty, with $\mathrm{poly}(|\mathfrak{G}|)$ working time.*

The whole algorithm described in this section works in $\mathrm{poly}(n, 2^d, n^b) = \mathrm{poly}(\|\Gamma \to C\|, 2^{\mathrm{ord}(\Gamma\to C)}, \|\Gamma \to C\|^{\mathrm{b}(\Gamma\to C)})$ time, as required in Theorem 4.

6 Proof of Theorem 18 (Pentus' Construction Revisited)

In our algorithm, described in Section 5, we use Pentus' polynomial-size context free grammar, that generates all proof nets, as a black box: we need only Theorem 18 itself, not the details of the construction in its proof. However, Pentus [35] doesn't explicitly formulate this theorem, but rather gives it as side-effect of the construction for checking *existence* of a proof net (e.g., non-emptiness of the context free language). The latter is, unfortunately, not sufficient for our needs. Moreover, we use slightly different complexity parameters. Therefore, and also in order to make our paper logically self-contained, in this section we redisplay Pentus' construction in more detail. In other words, here we prove Theorem 18.

Pentus' idea for seeking proof nets is based on dynamic programming. In $\Omega_{\Gamma\to C}$, connective and literal occurrences alternate: $c_1, \ell_1, c_2, \ell_2, \ldots, c_n, \ell_n$. Consider triples of the form (i, j, k), where $1 \leq i \leq j \leq k \leq n$.

▶ **Definition 22.** An (i, j, k)-*segment* $\tilde{\mathcal{E}}$ is a planar pairing of literals from $\{\ell_i, \ldots, \ell_{k-1}\}$ such that in every region created by $\tilde{\mathcal{E}}$ there exists a unique $\parr$ or $\diamond$ occurrence from $\{c_i, \ldots, c_k\}$ that belongs to this region, and, in particular, this occurrence for the outer (infinite) region is c_j, called the *open par*. If $k = j = i$, then $\tilde{\mathcal{E}}$ is empty, and c_i should be a $\parr$ or $\diamond$ occurrence.

For each $\tilde{\mathcal{E}}$ we construct the corresponding $\tilde{\mathcal{A}}$ that connects each $\otimes$ occurrence to the only $\parr$ or $\diamond$ in the same region; for the outer region, it uses the open par.

▶ **Definition 23.** An (i, j, k)-segment $\tilde{\mathcal{E}}$ is *correct,* iff the graph $\tilde{\mathcal{A}} \cup \prec$ is acyclic.

For each (i,j,k)-segment, in the non-terminals of the grammar we keep a small amount of information, which we call the *profile* of the segment and that is sufficient to construct bigger segments (and, finally, the whole proof net) from smaller ones.

▶ **Definition 24.** A $\otimes$ occurrence is called *dominant,* iff it is not immediately dominated (in the $\prec$ preorder) by another $\otimes$ occurrence. For each $\otimes$ occurrence c there exists a unique dominant $\otimes$ occurrence $\tau(c)$ such that $\tau(c) \succeq c$ and on the $\prec$ path from c to $\tau(c)$ all occurrences are $\otimes$ occurrences.

Let's call two $\otimes$ occurrences *equivalent,* $c \approx c'$, if $\tau(c) = \tau(c')$. Equivalent $\otimes$ occurrences form *clusters;* from each cluster we pick a unique representative, the $\prec$-maximal occurrence $\tau(c)$. By $\precsim$ we denote the transitive closure of $\prec \cup \approx$: $c \precsim c'$ means that there is a path from c to c' that goes along $\prec$ and also could go in the inverse direction, but only from $\otimes$ to $\otimes$ with no $\parr$ or $\diamond$ in between.

▶ **Lemma 25.** *For an (i,j,k)-segment $\tilde{\mathcal{E}}$, the graph $\tilde{\mathcal{A}} \cup \prec$ is acyclic iff any cycle in $\tilde{\mathcal{A}} \cup \precsim$ is a trivial $\approx$-cycle in a cluster, and similarly for a proof structre $\mathcal{E}$ and the graphs $\mathcal{A} \cup \prec$ and $\mathcal{A} \cup \precsim$.*

Proof. Pentus proves this lemma by a topological argument. If $\tilde{\mathcal{A}} \cup \precsim$ has non-trivial cycles, take a simple cycle (i.e. a cycle where no vertex appears twice) that embraces the smallest area. If this cycle includes a link from c to d where $c \approx d$ and $c \succ d$, then consider the $\tilde{\mathcal{A}}$ link that goes from c. This link should go *inside* the cycle, and, continuing by this link, one could construct a new cycle with a smaller area embraced. Contradiction. The other direction is trivial, since every cycle in $\tilde{\mathcal{A}} \cup \prec$ is a non-trivial cycle in $\tilde{\mathcal{A}} \cup \precsim$. ◀

In view of this lemma we can now use $\precsim$ instead of $\prec$ in the correctness (acyclicity) criteria for proof nets and (i,j,k)-segments.

▶ **Definition 26.** For a connective occurrence c_i let V_i be the set of all dominant $\otimes$ occurrences on the $\prec$ path from c_i to the root of the parse tree.

Since each dominant $\otimes$ marks a point of alternation between $\otimes$ and $\parr$ (or $\diamond$, on the top level), and the number of such alternations is bounded by d, we have $|V_i| \leq d$ for any i.

▶ **Definition 27.** The profile of an (i,j,k)-segment $\tilde{\mathcal{E}}$, denoted by R, is the restriction of the transitive closure of $\tilde{\mathcal{A}} \cup \precsim$ to the set $V_i \cup V_j \cup V_k$ that is forced to be irreflexive (in other words, we remove trivial $\approx$-cycles). An (i,j,k)-*profile* is an arbitrary transitive irreflexive relation on $V_i \cup V_j \cup V_k$.

▶ **Lemma 28.** *The number of different (i,j,k)-profiles is* $\mathrm{poly}(2^d)$.

Proof. Let $|V_i| = d_1$, $|V_j| = d_2$, $|V_k| = d_3$ (these three numbers are not greater than d). Each profile includes three chains, Q_i, Q_j, and Q_k, and it remains to count the number of possible connections between them. Due to transitivity, if a vertex in V_i is connected to a vertex in V_j, then it is also connected to all greater vertices. Now we represent elements of V_j as d_2 white balls, and put d_1 black balls between them. The i-th black ball is located in such a place that the i-th vertex of V_i is connected to all vertices of V_j that are greater than the position of the i-th ball, and only to them. Due to transitivity, the order of black balls is the same as Q_i. The number of possible distributions of white and black balls is $\binom{d_1+d_2}{d_1} < 2^{d_1+d_2} \leq 2^{2d}$. Doing the same for all 6 pairs of 3 chains, we get the estimation $(2^{2d})^6 = (2^d)^{12} = \mathrm{poly}(2^d)$ for the number of (i,j,k)-profiles. ◀

Now we define the context free grammar $\mathfrak{G}_1$. Non-terminal symbols of this grammar include the starting symbol S and symbols $F_{i,j,k,R}$ for any triple (i,j,k) $(1 \leq i \leq j \leq k \leq n)$ and any (i,j,k)-profile R. The meaning of these non-terminals is in the following statement, which will be proved by induction after we present the rules of $\mathfrak{G}_1$: a word w is derivable from $F_{i,j,k,R}$ iff $w = c(\tilde{\mathcal{E}})$ for a correct (i,j,k)-segment $\tilde{\mathcal{E}}$ with profile R; a word w is derivable from S iff $w = c(\mathcal{E})$ for some proof net $\mathcal{E}$. (Codes of (i,j,k)-segments are defined in the same way as codes of proof structures, as involutive permutations of $e_i, \ldots, e_{k-1}$.)

For the induction base case, $i = j = k$, we take only those values of i such that c_i is a $\parr$ or $\diamond$ occurrence, and denote by Q_i the $\prec$ relation restricted to V_i (this is the trivial profile of an empty (i,i,i)-segment); Q_i is always acyclic, and an isolated $\parr$ or $\diamond$ occurrence c_i is always a correct (i,i,i)-segment (with an empty $\tilde{\mathcal{E}}$), and c_i is its open par. Now for each $\parr$ or $\diamond$ occurrence c_i we add the following rule to the grammar (this is a ε-rule, the right-hand side is empty):

$$F_{i,i,i,Q_i} \Rightarrow .$$

Next, consider the non-trivial situation, where $i < k$. The difference $k - i$ should be even, otherwise there couldn't exist a literal pairing $\tilde{\mathcal{E}}$. Moreover, if both c_i and c_k are $\parr$ or $\diamond$ occurrences, a correct (i,j,k)-segment couldn't exist either, since in the outer region we have at least two $\parr$ or $\diamond$ occurrences, namely, c_i and c_k. Therefore, we include rules for $F_{i,j,k,R}$ only if $k - i$ is even and at least one of c_i and c_k should be $\otimes$. Let it be c_i (Pentus' *situation of the first kind*). The c_k case (Pentus' *situation of the second kind*) is handled symmetrically.

We take the leftmost literal occurrence, ℓ_i, and find all possible occurrences among $\ell_{i+1}, \ldots, \ell_{k-1}$ that could be connected to ℓ_i (i.e., if ℓ_i is an occurrence of q, we seek $\bar{q}$, and vice versa). For each such occurrence, ℓ_{h-1}, we consider two triples, $(i+1, j', h-1)$ and (h,j,k), and all possible $(i+1,j',h-1)$- and (h,j,k)-profiles, R_1 and R_2, respectively. For each such pair, R_1 and R_2, we consider the transitive closure of the following relation: $R_1 \cup R_2 \cup Q_i \cup \{\langle \tau(c_i), d\rangle \mid d \in V_j\}$. If it is irreflexive (acyclic), its restriction to $V_i \cup V_j \cup V_k$, denoted by R, will become a profile of an (i,j,k)-segment. For this, we add the following rule to the grammar:

$$F_{i,j,k,R} \Rightarrow e_{h-1}\, F_{i+1,j',h-1,R_1}\, e_i\, F_{h,j,k,R_2}.$$

▶ **Lemma 29.** *In this grammar, a word w can be derived from $F_{i,j,k,R}$ iff $w = c(\tilde{\mathcal{E}})$ for some (i,j,k)-segment $\mathcal{E}$ with profile R.*

Proof. Proceed by induction on $k - i$. The base case ($i = j = k$) was considered above.

For **the "only if" part,** let w be derived by a rule for the first kind (the second kind is symmetric). Then $w = e_{h-1}w_1e_iw_2$, and by induction hypothesis w_1 and w_2 encode $(i+1,j',h-1)$- and (h,j,k)-segments with profiles R_1 and R_2 respectively. The word w encodes an (i,j,k)-segment, and it remains to show that this segment is correct and its profile is R. For this new segment, $\tilde{\mathcal{A}} = \tilde{\mathcal{A}}_1 \cup \tilde{\mathcal{A}}_2 \cup \langle c_i, c_j\rangle$. Suppose there is a non-trivial cycle in $\tilde{\mathcal{A}} \cup \precsim$. Since all cycles in $\tilde{\mathcal{A}}_1 \cup \precsim$ and $\tilde{\mathcal{A}}_2 \cup \precsim$ are trivial, this cycle should either include links from both $\tilde{\mathcal{A}}_1$ and $\tilde{\mathcal{A}}_2$ or use the new $\langle c_i, c_j\rangle$ connection (or both). The cycle, however, cannot cross $\tilde{\mathcal{E}}$ links, therefore the only way of "legally crossing the border" between segments is by going through $\otimes$ occurrences that dominate c_i, c_{i+1}, c_{h-1}, c_h, or c_k. We can assume that these "border crossing points" are dominant $\otimes$ occurrences (otherwise we can add a $\approx$-detour to the cycle). Then the cycle is actually a concatenation of parts of the following three kinds: (1) connecting vertices of $V_{i+1} \cup V_{h-1}$; (2) connecting vertices of

$V_h \cup V_k$; (3) connecting $\tau(c_i)$, via c_j, to a vertex d of V_j. In this case, our cycle induces a cycle in $R_1 \cup R_2 \cup Q_i \cup \{\langle \tau(c_i), d\rangle \mid d \in V_j\}$, which is impossible by definition.

It remains to show that R is the profile of the newly constructed segment. Indeed, R is a binary relation on $V_i \cup V_j \cup V_k$ and is included in the transitive closure of $\tilde{\mathcal{A}} \cup \precsim$, therefore R is a subrelation of the profile. On the other hand, if there is a pair $\langle c, d\rangle$ in the profile, then there is a path from c to d and, as shown above, it can be split into parts of kinds (1), (2), and (3). Thus, $\langle c, d\rangle \in R$, and therefore R coincides with the profile.

For **the "if" part,** if c_i in an (i, j, k)-segment is a $\otimes$ occurrence, consider the $\tilde{\mathcal{E}}$ link from the literal occurrence ℓ_i. It splits the segment into two ones. For each of them, by induction hypothesis, we generate their codes from $F_{i+1,j',h-1,R_1}$ and F_{h,j,k,R_2} respectively, and then apply the rule to generate the code of the original segment. Situations of the second kind, where c_k is a $\otimes$ occurrence, are handled symmetrically. ◀

Finally, we add rules for the starting symbol. These rules are analogous to the rules for situations of the second kind. Take ℓ_n and find all possible occurrences among $\ell_1, \ldots, \ell_{n-1}$ that could be connected to it. For each such occurrence ℓ_h and any pair of $(0, 0, h)$- and $(h+1, j', n)$-profiles, R_1 and R_2, respectively (in the first segment $j = 0$, since in the whole proof net the open par should be the leftmost occurrence of $\diamond$), consider the transitive closure of $R_1 \cup R_2$. If it is irreflexive, then we add the following rule to the grammar:

$$S \Rightarrow F_{0,0,h,R_1}\ e_n\ F_{h+1,j',n,R_2}\ e_h.$$

▶ **Lemma 30.** *A word w can be derived from S iff $w = c(\tilde{\mathcal{E}})$ for some proof net $\mathcal{E}$.*

Proof. Analogous to the previous lemma. ◀

This lemma shows that we've constructed a grammar that generates P_1. Now to finish the proof of Theorem 18 it remains to establish complexity bounds. The number of non-terminal symbols is bounded by $n^3 \cdot K + 1$, where K is the maximal number of (i, j, k)-profiles. Since each rule has length at most 5 (1 non-terminal on the left and 4 symbols on the right), $|\mathfrak{G}_1|$ is bounded by $5(n^3 \cdot K + 1)$, and, since K is poly(2^d) (Lemma 28), $|\mathfrak{G}_1|$ is poly$(2^d, n)$. Clearly, the procedure that generates $\mathfrak{G}_1$ from the original sequent is also polynomial in running time: acyclicity checks for each rule are performed in poly(n) time, and the number of rules is poly$(2^d, n)$.

7 Conclusions and Future Work

In this paper, we've presented an algorithm for provability in the Lambek calculus with brackets. Our algorithm runs in polynomial time w.r.t. the size of the input sequent, if its order and bracket nesting depth are bounded. Our new result for bracket modalities is non-trivial, and we address it with a combination of proof nets and finite automata techniques.

We summarize some questions raised for future research. First, Pentus [35] also presents a parsing procedure for Lambek categorial grammars. In a Lambek grammar, several types can be assigned to one word, which adds an extra level of non-determinism. Our intention is to develop an efficient parsing procedure for grammars with brackets. Second, the problem whether $\mathbf{Lb}^*$-grammars define exactly context free languages is still open (the counter-example by Fadda and Morrill [11] jeopardises Jäger's claim). Third, in our calculus we allow empty antecedents. We are going to modify our algorithm for the bracketed extension of the original Lambek calculus, using a modified notion of proof nets (see for example [22][20]). A more

general question is to extend the algorithm to other enrichments of the Lambek calculus (see, for example, [28]), keeping polynomiality, if possible. Notice that some of these enrichments are generally undecidable [18], so it is interesting to find feasible bounded fragments.

Acknowledgements. The authors are grateful to Mati Pentus for in-depth comments on his algorithm [35].

References

1 E. Aarts. Proving theorems of second order Lambek calculus in polynomial time. *Studia Logica*, 53:373–387, 1994.

2 E. Aarts and K. Trautwein. Non-associative Lambek categorial grammar in polynomial time. *Math. Logic Quart.*, 41:476–484, 1995.

3 V. M. Abrusci. Non-commutative proof nets. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*. Cambridge University Press, 1995.

4 K. Ajdukiewicz. Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27, 1935.

5 Y. Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.

6 W. Buszkowski. Type logics in grammar. In *Trends in Logic: 50 Years of Studia Logica*, pages 337–382. Springer, 2003.

7 B. Carpenter. *Type-Logical Semantics*. MIT Press, Cambridge, MA, 1997.

8 V. Danos and L. Regnier. The structure of multiplicatives. *Arch. Math. Log.*, 28:181–203, 1989.

9 Ph. de Groote. A dynamic programming approach to categorial deduction. In H. Ganzinger, editor, *Proc. CADE 1999*, volume 1632 of *Lect. Notes Comput. Sci.*, pages 1–15. Springer, 1999.

10 Ph. de Groote. The non-associative Lambek calculus with product in polynomial time. In *Proc. TABLEAUX 1999*, pages 128–139. Springer, 1999.

11 M. Fadda and G. Morrill. The Lambek calculus with brackets. In *Language and Grammar: Studies in Mathematical Linguistics and Natural Language*, pages 113–128. CSLI, Jan 2005.

12 T. Fowler. Efficient parsing with the product-free Lambek calculus. In *Proc. COLING 2008*, 2008.

13 T. Fowler. A polynomial time algorithm for parsing with the bounded order Lambek calculus. In *Proc. MoL 2009*, 2009.

14 S. Ginsburg. *The mathematical theory of context-free languages*. McGraw-Hill, 1966.

15 J.-Y. Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

16 G. Jäger. On the generative capacity of multi-modal categorial grammars. *Research on Language and Computation*, 1(1–2):105–125, 2003.

17 G. Jäger. *Anaphora and Type Logical Grammar*, volume 24 of *Trends in Logic – Studia Logica Library*. Springer, Dordrecht, 2005.

18 M. Kanovich, S. Kuznetsov, and A. Scedrov. Undecidability of the Lambek calculus with a relevant modality. In *Proc. Formal Grammar 2015 and 2016*, volume 9804 of *Lect. Notes Comput. Sci.*, pages 240–256. Springer, 2016.

19 N. Kurtonina. *Frames and labels. A modal analysis of categorial inference*. PhD thesis, Universiteit Utrecht, ILLC, Amsterdam, 1995.

20 S. Kuznetsov. Lambek grammars with one division and one primitive type. *Log. J. IGPL*, 20(1):207–221, 2012.

21 S. L. Kuznetsov. On translating Lambek grammars with one division into context-free grammars. *Proc. Steklov Inst. Math.*, 294:129–138, 2016.

22 F. Lamarche and C. Retoré. Proof nets for the Lambek calculus—an overview. In V. M. Abrusci and C. Casadio, editors, *Proofs and Linguistic Categories, Proc. 1996 Roma Workshop*, pages 241–262. CLUEB, 1996.

23 J. Lambek. The mathematics of sentence structure. *Amer. Math. Mon.*, 65:154–170, 1958.

24 J. Lambek. On the calculus of syntactic types. In *Structure of Language and Its Mathematical Aspects*, volume 12 of *Proc. Symposia Appl. Math.*, pages 166–178. AMS, 1961.

25 M. Moortgat. Multimodal linguistic inference. *J. Log. Lang. Inform.*, 5(3, 4):349–385, 1996.

26 R. Moot and C. Retoré. *The Logic of Categorial Grammars: A Deductive Account of Natural Language Syntax and Semantics*. Springer, Heidelberg, 2012.

27 G. Morrill. Categorial formalisation of relativisation: pied piping, islands, and extraction sites. Technical Report LSI-92-23-R, Universitat Politècnica de Catalunya, 1992.

28 G. V. Morrill. *Categorial Grammar: Logical Syntax, Semantics, and Processing*. Oxford University Press, 2011.

29 M. Nagayama and M. Okada. A graph-theoretic characterization theorem for multiplicative fragment of non-commutative linear logic. *Theor. Comput. Sci.*, 294:551–573, 2003.

30 G. Penn. A graph-theoretic approach to sequent derivability in the Lambek calculus. *Electr. Notes Theor. Comput. Sci.*, 53, 2002.

31 M. Pentus. Lambek grammars are context-free. In *Proc. LICS 1993*, pages 430–433, Montreal, 1993.

32 M. Pentus. Models for the Lambek calculus. *Ann. Pure Appl. Log.*, 75(1–2):179–213, 1995.

33 M. Pentus. *Free monoid completeness of the Lambek calculus allowing empty premises*, volume 12 of *Lecture Notes in Logic*, pages 171–209. Springer-Verlag, Berlin, 1998.

34 M. Pentus. Lambek calculus is NP-complete. *Theor. Comput. Sci.*, 357(1):186–201, 2006.

35 M. Pentus. A polynomial-time algorithm for Lambek grammars of bounded order. *Linguistic Analysis*, 36(1–4):441–471, 2010.

36 Yu. Savateev. Unidirectional Lambek grammars in polynomial time. *Theory Comput. Syst.*, 46(4):662–672, 2010.

37 K. Versmissen. *Grammatical composition: modes, models, modalities*. PhD thesis, OTS Utrecht, 1996.

38 D. N. Yetter. Quantales and (noncommutative) linear logic. *J. Symb. Log.*, 55(1):41–64, 1990.

Polynomial Running Times for Polynomial-Time Oracle Machines*†

Akitoshi Kawamura[1] and Florian Steinberg[2]

1 **Graduate School of Arts and Sciences, University of Tokyo, Tokyo, Japan**
kawamura@graco.c.u-tokyo.ac.jp
2 **Department of Mathematics, Technische Universität Darmstadt, Darmstadt, Germany**
steinberg@mathematik.tu-darmstadt.de

Abstract

This paper introduces a more restrictive notion of feasibility of functionals on Baire space than the established one from second-order complexity theory. Thereby making it possible to consider functions on the natural numbers as running times of oracle Turing machines and avoiding second-order polynomials, which are notoriously difficult to handle. Furthermore, all machines that witness this stronger kind of feasibility can be clocked and the different traditions of treating partial functionals from computable analysis and second-order complexity theory are equated in a precise sense. The new notion is named 'strong polynomial-time computability', and proven to be a strictly stronger requirement than polynomial-time computability. It is proven that within the framework for complexity of operators from analysis introduced by Kawamura and Cook the classes of strongly polynomial-time computable functionals and polynomial-time computable functionals coincide.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes

Keywords and phrases second-order complexity, oracle Turing machine, computable analysis, second-order polynomial, computational complexity of partial functionals

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.23

1 Introduction

Modern applications of second-order complexity theory almost exclusively use time-restricted oracle Turing machines to define and argue about the class of polynomial-time computable functionals [14, 7, 4, 3, 5, 12, 16, etc.]. The acceptance of this model of computation goes back to a result by Kapron and Cook [6] that characterizes the class of basic feasible functionals introduced by Mehlhorn [15].

There are several reasons for the popularity of this model of computation. Firstly, it intuitively reflects what a programmer would require of an efficient program if oracle Turing machines are interpreted as programs with subroutine calls. I.e. the time taken to evaluate the subroutine is not counted towards the time consumption (the oracle query takes one time step) and if the result is complicated the machine is given more time for further operations. Secondly, it is superficially quite close to classical polynomial-time computability: There is

* A longer version of the paper is available on the arXiv [13], https://arxiv.org/abs/1704.01405.

† This research was partly supported by the Japan Society for the Promotion of Science (JSPS) KAKENHI JP26700001 and the Core-to-Core Program (Advanced Research Networks).

a type of functions that take sizes of the inputs and return an allowed number of steps. A subclass of these functions are considered polynomial, or 'fast' running times.

On closer inspection, however, the second-order framework introduces a whole bunch of new difficulties: Running times of oracle Turing machines, and also the functions that are considered polynomial running times, are functions of type $\mathbb{N}^{\mathbb{N}} \times \mathbb{N} \to \mathbb{N}$. These so-called second-order polynomials are a lot less well-behaved than their first-order counterparts. There are no normal-form theorems, structural induction turns out to be complicated, there is no established notion of degree and so on [10, 11]. Even worse: Second-order polynomials turn out to not be time-constructible [16].

The framework introduced by Kawamura and Cook [8] addresses this problem by restricting to length-monotone string functions, thereby forcing time-constructibility of second-order polynomials. However, it has been argued that the restriction to length-monotone string functions seems to be an unnatural one in practice [16] and that it is too restrictive to reflect some situations from practice [1]. Thus, this paper investigates different solutions to the above problems.

The content of this paper

The first part of the paper introduces the notion of polynomial-time computability from second-order complexity theory. While usually only total functionals are considered, it introduces two possible generalizations of the definition to partial functionals. The traditions of how to handle partial functionals differ a lot between computable analysis and second-order complexity theory. The two corresponding notions are proven to be distinct. This can be considered to be a very strong version of the statement that second-order polynomials are not time-constructible. Finally, it is proven that in the most important example of use of an intermediate of the two conventions, namely the framework for complexity of operators in analysis as introduced by Kawamura and Cook, could have equivalently used the convention from second-order complexity theory.

The second part of the paper presents a restriction on the behavior of oracle machines such that use of running times of higher type is not necessary anymore. It proves that the corresponding class of functionals, which are named 'strongly polynomial-time computable functionals', is a subclass of the class of polynomial-time computable functionals as defined in second-order complexity theory. It provides an example of a functional which is polynomial-time computable but not strongly polynomial-time computable. The example is not a natural, but it is pointed out without a proof that there is a candidate for a more natural example. Finally the paper presents some evidence that strong polynomial-time computability is more compatible with partial functionals and proves that within the framework for complexity of operators in analysis introduced by Kawamura and Cook, it is equivalent to polynomial-time computability.

An extended version of this paper can be found on the arXiv [13].

Conventions

Fix the finite alphabet $\Sigma := \{\mathtt{0}, \mathtt{1}\}$ and let Σ^* denote the set of finite binary strings. Elements of Σ^* are denoted by $\mathbf{a}$, $\mathbf{b}$, ... The set of non-negative integers is denoted by $\mathbb{N}$ and its elements by n, m, ... We identify the Baire space with the set $\mathcal{B} := (\Sigma^*)^{\Sigma^*}$ of string functions. Elements of $\mathcal{B}$ are denoted as φ, ψ, ... We assume the reader to be familiar with the notions of computability and complexity for elements of the Baire space introduced via Turing machines.

To compute functions on the Baire space, this paper uses oracle Turing machines: An oracle Turing machine $M^?$ is a Turing machine that has an additional oracle query tape and an oracle query state. The oracle slot may be occupied by any string function $\varphi \in \mathcal{B}$. If the computation of $M^?$ with oracle φ and input $\mathbf{a}$, also referred to as the run of M^φ on $\mathbf{a}$, enters the query state, the content of the oracle query tape, say $\mathbf{a}$, is replaced with the value $\varphi(\mathbf{a})$. For measuring time consumption, overwriting $\mathbf{a}$ with $\varphi(\mathbf{a})$ is considered to be done in one time step and the reading/writing head is not moved. The outcome of the run of $M^?$ with oracle φ on input $\mathbf{a}$ is denoted by $M^\varphi(\mathbf{a})$ and the time this computation takes by $\mathrm{time}_{M^\varphi}(\mathbf{a})$. For $A \subseteq \mathcal{B}$ an functional $F\colon A \to \mathcal{B}$ reps. a functional $\tilde{F}\colon A \times \Sigma^* \to \Sigma^*$ if for all $\varphi \in A$ and $\mathbf{a} \in \Sigma^*$ it holds that $F(\varphi)(\mathbf{a}) = M^\varphi(\mathbf{a})$ resp. $\tilde{F}(\varphi, \mathbf{a}) = M^\varphi(\mathbf{a})$.

2 Second-order complexity theory and relativization

We usually consider functionals to be of objects of type $\mathcal{B} \to \mathcal{B}$. This section is an exception as for complexity considerations it is more natural to consider oracle Turing machines to compute objects of type $\mathcal{B} \times \Sigma^* \to \Sigma^*$. Both the string and the oracle are considered input. A meaningful bound on the number of steps a machine takes should be allowed to depend on the sizes of the inputs. The size of a binary string $\mathbf{a}$ is its binary length $|\mathbf{a}| \in \mathbb{N}$. But also computations on 'big' oracles φ should be granted more time. The size of an oracle is not a natural number anymore, but a function on the natural numbers:

▶ **Definition 1.** For a string function $\varphi \in \mathcal{B}$, define its **size function** $|\varphi| : \mathbb{N} \to \mathbb{N}$ by

$$|\varphi|(n) := \max\{|\varphi(\mathbf{a})| \mid |\mathbf{a}| \leq n\}.$$

Thus, running times are objects of the type $T\colon \mathbb{N}^\mathbb{N} \times \mathbb{N} \to \mathbb{N}$: Such a function T is a running time for an oracle Turing machine $M^?$ if for any oracle φ and string $\mathbf{a}$, the run of M^φ on input $\mathbf{a}$ terminates within $T(|\varphi|, |\mathbf{a}|)$ steps. Note that it is not a priori clear what running times should be considered polynomial. The class of second-order polynomials is the smallest class of functions of type $\mathbb{N}^\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that:

- All of the functions $(l, n) \mapsto p(n)$ are contained, where p is a polynomial with natural numbers as coefficients.

And which is closed under the following operations:

- Whenever P and Q are contained, then so is their point-wise sum $P + Q$.
- Whenever P and Q are contained, then so is their point-wise product $P \cdot Q$.
- Whenever P is contained then so is the function P^+ defined by

$$P^+(l, n) := l(P(l, n)).$$

▶ **Definition 2.** A functional on Baire space is called **polynomial-time computable** if it is computed by an oracle Turing machine $M^?$ whose running time is bounded by a second-order polynomial. I.e. such that

$$\forall \varphi \in \mathcal{B}, \forall \mathbf{a} \in \Sigma^*\colon \mathrm{time}_{M^\varphi}(\mathbf{a}) \leq P(|\varphi|, |\mathbf{a}|).$$

The above definition is based on a characterization by Kapron and Cook of the class of basic feasible functionals originally introduced by Mehlhorn. Note that it is not obvious from the above definition that the class of polynomial-time computable functionals is closed under composition. To prove closure we need the following two closure properties of the set of second-order polynomials:

▶ **Lemma 3.** *Whenever P and Q are second-order polynomials, then so are*

$$(l,n) \mapsto P(Q(l,\cdot),n) \quad \text{and} \quad (l,n) \mapsto P(l,Q(l,n)).$$

The proof can be done via a tedious but straightforward induction on the term structure of second-order polynomials (an alternative proof is given in Appendix A).

▶ **Proposition 4.** *Let $F\colon \mathcal{B} \to \mathcal{B}$ and $G\colon \mathcal{B} \to \mathcal{B}$ be functionals on Baire space that can be computed within time P resp. Q. Then $F \circ G$ can be computed in time*

$$(l,n) \mapsto C \cdot (P(Q(l,\cdot),n) + Q(l,P(Q(l,\cdot),n)) \cdot P(Q(l,\cdot),n))$$

for some $C \in \mathbb{N}$. In particular, the polynomial-time computable functionals are closed under composition.

Proof (Idea). The running time bound can be obtained by combining machines that compute F and G into one that computes the composition and estimating the time this machine takes. ◀

Another property of polynomial-time computable functionals that should be mentioned is that they preserve the class of polynomial-time computable functions. This can easily be checked by combining the program of a polynomial-time machine computing the function with the program of a polynomial-time oracle Turing machine computing the functional.

Second-order polynomials were introduced as functions of type $\mathbb{N}^{\mathbb{N}} \times \mathbb{N} \to \mathbb{N}$. This is natural since they are considered running times. However, it also regularly leads to difficulties: It is not clear how to decide equality of two second-order polynomials from given construction procedures. The reader may for instance try to prove that the inequality $P \neq Q$ of two second-order polynomials as functions implies that also $P^+ \neq Q^+$. While a proof for the general case is not known to the authors, the proof becomes easy in the case where $P \neq Q$ is realized by an injective function argument. Note, that while it is not an unreasonable idea to restrict the domain of the second-order polynomials, it should at least contain all (not necessarily strictly) monotone functions, as these show up as length functions of string functions. Just like for the general case, a proof of the above if the inequality is realized by a monotone function is not known to the authors.

2.1 Relativization

Second-order complexity theory usually only considers total functionals. However, the application we are most interested in is real complexity theory, which stems from computable analysis. In computable analysis, computations on continuous structures are carried out by encoding the objects by string functions and operating on these. In this process, partial functionals are used. To see how, recall some notions from computable analysis:

▶ **Definition 5.** A **representation** ξ of a space X is a partial surjective mapping $\xi : \mathcal{B} \to X$.

An element of $\xi^{-1}(x)$ is called a **ξ-name** of x or just a **name**, if the representation is clear from the context. A pair $\mathbf{X} = (X, \xi_{\mathbf{X}})$ of a set and a representations of that set is called a represented space.

Computations on represented spaces can be carried out by operating on names:

▶ **Definition 6.** Let $f : \mathbf{X} \to \mathbf{Y}$ be a function between represented spaces. A partial functional $F :\subseteq \mathcal{B} \to \mathcal{B}$ is called a **realizer** of f if it translates $\xi_{\mathbf{X}}$-names of x to $\xi_{\mathbf{Y}}$-names of $f(x)$, that is if

$$\forall \varphi \in \mathrm{dom}(\xi_{\mathbf{X}}) : \xi_{\mathbf{Y}}(F(\varphi)) = f(\xi_{\mathbf{X}}(\varphi)).$$

A function is called **computable** if there is an oracle machine $M^?$ that computes a realizer F in the sense that $\forall \varphi \in \mathrm{dom}(\xi_{\mathbf{X}}) : M^\varphi = F(\varphi)$. Here, as is tradition in computable analysis, no assumptions are made about the behavior of the realizer outside of the domain of the representation. The characterization of polynomial-time computability of a functional by Kapron and Cook can straightforwardly be relaxed to fit this tradition.

▶ **Definition 7.** Let $A \subseteq \mathcal{B}$. We say that an oracle Turing machine $M^?$ runs in **A-restricted polynomial-time** if there exists a second-order polynomial P such that for each oracle φ from A and string $\mathbf{a}$ the computation of $M^\varphi(\mathbf{a})$ takes at most $P(|\varphi|, |\mathbf{a}|)$ steps. I.e. $\forall \varphi \in A, \forall \mathbf{a} \in \Sigma^* \colon \mathrm{time}_{M^\varphi}(\mathbf{a}) \leq P(|\varphi|, |\mathbf{a}|)$. We denote the set of functionals $F \colon A \to \mathcal{B}$ such that there is a machine computing F in A-restricted polynomial time by $\mathrm{P}(A)$.

There are two main examples of this definition covertly showing up in literature:

▶ **Example 8** (relativization)**.** Oracle machines are used in classical complexity theory to talk about polynomial-time computability of a string function $\varphi \colon \Sigma^* \to \Sigma^*$ relative to some oracle $\psi : \Sigma^* \to \{\mathtt{0}, \mathtt{1}\}$ interpreted as a subset of the strings. Under the assumption that ψ only retruns `0` or `1`, one can check that the following are equivalent:

- φ is polynomial-time computable relative to ψ.
- The constant functional returning φ is $\{\psi\}$-restricted polynomial-time computable.

Thus, the definition provides a generalization of relative polynomial-time computability, which is the reason for the name of this chapter.

The second example is Kawamura and Cook's framework for complexity for operators in analysis. Recall that Kawamura and Cook introduce the following subclass of Baire space:

▶ **Definition 9** ([8])**.** A string function $\varphi \in \mathcal{B}$ is called **length-monotone** if for all strings $\mathbf{a}$ and $\mathbf{b}$ it holds that $|\mathbf{a}| \leq |\mathbf{b}|$ implies $|\varphi(\mathbf{a})| \leq |\varphi(\mathbf{b})|$. The set of all length-monotone string functions is denoted by Σ^{**}.

Polynomial-time computability of functionals from Σ^{**} to Σ^{**} is then defined as Σ^{**}-restricted polynomial-time computability. (Of course it is not referred to by this name, but the definitions are identical.)

The tradition in second-order complexity theory is to impose the running time requirement independently of the domain of the functional.

▶ **Definition 10.** For $A \subseteq \mathcal{B}$ denote the class of all functionals $F \colon A \to \mathcal{B}$ that have a polynomial-time computable extension to all of Baire space by $\mathrm{P}|_A$.

For a partial functional $F \colon A \subseteq \mathcal{B} \to \mathcal{B}$ there are now two approaches to define polynomial-time computability. On one hand, one could require that F is A-restricted polynomial-time computable, i.e., $F \in \mathrm{P}(A)$. On the other hand, one could use the more restrictive definition that F has a total polynomial-time computable extension, i.e., $F \in \mathrm{P}|_A$. The first definition fits into the tradition of computable analysis, where usually no assumptions about a realizer are made outside of the domain of the representation on the input side of the operator. The second definition is in the tradition of second-order complexity theory, where one usually only considers polynomial-time computability of total functionals.

Note, that this situation is already encountered in classical complexity theory: For a partial function $\varphi : \Omega \subseteq \Sigma^* \to \Sigma^*$, polynomial time computability can be defined by requiring the existence of a machine M that computes φ and a polynomial p such that either

$$\forall \mathbf{a} \in \Omega \colon \mathrm{time}_M(\mathbf{a}) \leq p(a) \quad \text{or} \quad \forall \mathbf{a} \in \Sigma^* \colon \mathrm{time}_M(\mathbf{a}) \leq p(a).$$

Due to the time-constructibility of polynomials, these choices lead to the same class of partial functions. This argument does not generalize to the second-order framework.

2.2 Incompatibility with relativization

Of course, the distinction between $\mathrm{P}(A)$ and $\mathrm{P}\,|_A$ only makes sense if these classes differ in general. Note that by definition $\mathrm{P}(A) \supseteq \mathrm{P}\,|_A$. Before we give the example that separates the classes, let us discuss why this result is not obvious. The basic idea is to consider the length function on the string functions. Any oracle machine that computes this function takes a minimum of 2^n steps on any input of length n and arbitrary oracle, as each query of length n has to be asked to guarantee correctness of the return value. On the other hand, the brute-force search computes the length function in about $2^n l(n)$ time steps. This means, that the length function becomes A-restricted polynomial-time computable if A is chosen as the set of string functions that have at least exponential length.

Why does this not provide a counterexample already? Unfortunately, the brute force search can be modified to detect names of subexponential length and abort the computation in time. Informally such an algorithm can be described as follows: 'Do a brute-force search, but abort as soon as you have to ask more than twice as many oracle queries as the length of the biggest return value you have found so far'. Such a machine does indeed compute the restriction of the length function on the exponentially growing functions while running in polynomial time for all inputs and returning something that differs from the length on the shorter functions (this is allowed since they are not in the domain).

Thus, the argument has to be more elaborate. Our solution is to delay the time until a big return values are guaranteed: The elements of A are only required to exhibit exponential growth on a sparse subset, i.e. $|\varphi|\,(g(n)) \geq 2^{g(n)}$, where g is a fast growing function. Note that if g does not grow fast enough, the trick above does still work. For instance for $g(n) = 2^n$, the following algorithm works: 'Do a brute-force search but abort as soon as you have to ask more queries than the square of the biggest return value you have found so far'. If g grows too fast the A-restricted polynomial-time computability may break down.

Fortunately the choice $g(n) = 2^{2^n}$ is a sweet spot: On one hand, due to the availability of length function iteration, it is still possible to extract a super exponential function from an element of the set therefore to make the brute-force algorithm work in A-restricted polynomial-time. On the other hand the above approach to compute a total extension does not work anymore and it becomes provable that no polynomial-time computable extension exists.

▶ **Theorem 11** (in general $\mathrm{P}\,|_A \subsetneq \mathrm{P}(A)$)**.** *There exist a set $A \subseteq \mathcal{B}$ and a functional $F : A \to \mathcal{B}$ such that F is A-restricted polynomial-time computable but has no total polynomial-time computable extension.*

Proof. Consider the set

$$A := \{\varphi \in \mathcal{B} \mid \forall n \in \mathbb{N} : |\varphi|\,(2^{2^n}) \geq 2^{2^{2^n}}\}$$

and the functional on A defined by

$$F : A \to \mathcal{B}, \quad F(\varphi)(\mathbf{a}) := 0^{|\varphi|(|\mathbf{a}|)}.$$

F is A-restricted polynomial-time computable. To see that this is true first note that $3(n+2) \geq 2^{\lceil \mathrm{lb}(\mathrm{lb}(3(n+2)))\rceil - 1} \in \mathbb{N}$ and $\mathrm{lb}(n)^2 \leq 3(n+2)$ (this is implied by the inequality $\ln(x) \leq \frac{x-1}{\sqrt{x}}$). Thus, for $\varphi \in A$ it holds that

$$\begin{aligned}|\varphi|\,(|\varphi|\,(3(n+2))) &\geq |\varphi|\,(|\varphi|\,(2^{2^{\lceil \mathrm{lb}(\mathrm{lb}(3(n+2)))\rceil - 1}})) \geq |\varphi|\,(2^{2^{2^{\lceil \mathrm{lb}(\mathrm{lb}(3(n+2)))\rceil - 1}}}) \\ &\geq 2^{2^{2^{2^{\lceil \mathrm{lb}(\mathrm{lb}(3(n+2)))\rceil - 1}}}} \geq 2^{2^{\sqrt{3(n+2)}}} \geq 2^n\end{aligned}$$

This means that a second-order polynomial provides sufficient time to find the value of $|\varphi|(|\mathbf{a}|)$ in A-restricted polynomial time using a brute-force search.

However, F does not have a total polynomial-time computable extension, as can be seen as follows: Towards a contradiction assume that there is an oracle Turing machine $M^?$ that computes such an extension in time bounded by some second-order polynomial P. For each $n \in \mathbb{N}$ define an oracle $\varphi_n \in A$. First define a sequence of functions $\varphi_{n,k} \in \mathcal{B}$. Let $\varphi_{n,0}$ be the constant function returning ε. To recursively define $\varphi_{n,k+1}$ follow the computation $M^{\varphi_{n,k}}(0^{2^{2^n}-1})$ and whenever a query $\mathbf{a}$ is asked such that $\mathrm{lb}(\mathrm{lb}(|\mathbf{a}|))$ is an integer, then check whether all other queries of this length have been asked before and were answered with an ε by $\varphi_{n,k}$. If this situation is encountered for some query $\mathbf{a}_k$, then set $\varphi_{n,k+1}(\mathbf{a}_k)$ to be the string of $2^{2^{2^m}}$ zeros, ignore the rest of the computation and set for $\varphi_{n,k+1}(\mathbf{b}) := \varphi_{n,k}(\mathbf{b})$ for all other strings. If such an $\mathbf{a}_k$ does not exist, then set $\varphi_{n,k+1} := \varphi_{n,k}$. The sequence $(\varphi_{n,k})_k$ converges in Baire space, as the sequence $\varphi_{n,k}(\mathbf{a})$ is either constantly ε or jumps to $0^{2^{|\mathbf{a}|}}$ at some point and remains constant afterwards. Let $\tilde{\varphi}_n$ be the limit. Since $M^?$ is a deterministic machine, the computations $M^{\tilde{\varphi}_n}(0^{2^{2^n}-1})$ and $M^{\varphi_{n,k}}(0^{2^{2^n}-1})$ are identical up until the query $\mathbf{a}_{k+1}$ is done. For the computation on oracle $\tilde{\varphi}_n$ to be finite, the sequence $(\mathbf{a}_k)_k$ must be finite. Let k_0 be bigger than the number of elements, then $\tilde{\varphi}_n = \varphi_{n,k_0}$. Let φ_n be the function that is identical to φ_{n,k_0} unless φ_{n,k_0} returns ε on all inputs of length 2^{2^m}. In this case not all the queries of this length were asked in the run of the machine $M^?$ on oracle φ_{n,k_0} and input $0^{2^{2^n}-1}$. Pick one query of length 2^{2^m} that was not asked and let φ_n return the string of $2^{2^{2^m}}$ zeros on this string. This guarantees that $\varphi_n \in A$.

Let ψ_n be the string function that coincides with φ_n on strings of length less or equal $2^{2^{n-1}}$ (and thus also on all strings of length less or equal $2^{2^n}-1$) and returns ε on bigger strings. Since the machine $M^?$ is deterministic for $M^{\varphi_n}(0^{2^{2^n}-1})$ and $M^{\psi_n}(0^{2^{2^n}-1})$ to differ it is necessary that an oracle query has been asked such that the answers of ψ_n and φ_n are distinct. The definition of φ_n makes sure that this does not happen before all queries of length 2^{2^n} have been posed. Each of these queries takes one time step, thus $\mathrm{time}_{M^{\psi_n}}(\mathbf{a}) \geq 2^{2^{2^n}}$ if the return values differ. If the machine runs identically on oracle φ_n and oracle ψ_n, then it has to ask each query of length $2^{2^n}-1$ to correctly compute the length (otherwise we may change the value in the query of length $2^{2^n}-1$ that was not asked). Thus, for all n

$$\mathrm{time}_{M^{\psi_n}}(0^{2^{2^n}-1}) \geq 2^{2^{2^n}-1}.$$

By the definition of ψ_n it holds that $|\psi_n|(k) \leq 2^{2^{2^{n-1}}}$ for all k. Note that whenever l is monotone and bounded by $m \in \mathbb{N}$, i.e. $l(k) \leq m$ for all $k \in \mathbb{N}$, then there exists a polynomial p such that

$$P(l,k) \leq \max\{p(m), p(k)\}.$$

Therefore, it holds for all n and appropriate $C, d \in \mathbb{N}$ that

$$P(|\psi_n|, 2^{2^n}-1) \leq \max\{C2^{d2^{2^{n-1}}}, p(2^{2^n}-1)\}.$$

Using that P is a running time of $M^?$ and the inequality from above obtain

$$2^{2^{2^n}-1} \leq \mathrm{time}_{M^{\varphi_n}}(0^{2^{2^n}-1}) = \mathrm{time}_{M^{\psi_n}}(0^{2^{2^n}-1}) \leq \max\{C2^{d2^{2^{n-1}}}, p(2^{2^n}-1)\}.$$

The maximum on the far right is assumed by the first term only for finitely many n: $2^{2^n}-1 \leq d2^{2^{n-1}} + \mathrm{lb}(C)$ is a quadratic inequality for $x := 2^{2^{n-1}}$ and the set where it is fulfilled can be specified explicitly. However, this implies that the left hand side is bounded by a polynomial in $2^{2^n}-1$ which is clearly not the case. A contradiction. ◀

2.3 A partial recovery

The previous section proved that for an arbitrary set $A \subseteq \mathcal{B}$, it can not be expected that every A-restricted polynomial-time computable functional has a total polynomial-time computable total extension. However, Σ^{**} is far from being an arbitrary set. Recall the following notion:

▶ **Definition 12.** Let A be a subset of $\mathcal{B}$. A mapping $R\colon \mathcal{B} \to A$ is called a **retraction** of $\mathcal{B}$ onto A, if for all $\varphi \in A$ it holds that $R(\varphi) = \varphi$.

A property of Σ^{**} that guarantees the existence of total polynomial-time computable extensions is the following:

▶ **Lemma 13.** *There is a polynomial-time computable retraction from $\mathcal{B}$ onto Σ^{**}.*

Proof. For a string $\mathbf{a}$ let $\mathbf{a}^{\leq n}$ denote its initial segment of length n (or the string itself if it has less than n bits). Consider the mapping

$$R(\varphi)(\mathbf{a}) := \varphi(\mathbf{a})^{\leq |\varphi(0^n)|} 0^{\max\{|\varphi(0^n)| - |\varphi(\mathbf{a})|, 0\}}.$$

This mapping is a polynomial-time computable retraction from $\mathcal{B}$ onto Σ^{**}. ◀

▶ **Theorem 14.** *Whenever there is a polynomial-time computable retraction from $\mathcal{B}$ onto A, then any A-restricted polynomial-time computable functional has a total polynomial-time computable extension. I.e.* $\mathrm{P}\,|_A = \mathrm{P}(A)$.

Proof (Idea). The proof of Proposition 4 that the composition of two polynomial-time computable functionals is polynomial-time computable remains valid if the assumptions are weakened to F being $G(\mathcal{B})$-restricted polynomial-time computable. Thus, the composition of the A-restricted polynomial-time computable functional with the retraction is polynomial-time computable. ◀

The previous two results directly entail the following:

▶ **Corollary 15** ($\mathrm{P}(\Sigma^{**}) = \mathrm{P}\,|_{\Sigma^{**}}$)**.** *A functional $F : \Sigma^{**} \to \mathcal{B}$ is polynomial-time computable in the sense of Kawamura and Cook if and only if it has a total polynomial-time computable extension.*

An alternative proof can be obtained by adding a clocks to machines. (More details about how to clock machines can be found in the proof of Theorem 25.)

3 Query dependent step restrictions

This section investigates a different approach to measuring the running time of an oracle machine. The alternative approach does not rely on higher order objects as running times. Recall that for a regular Turing machine the time function $\mathrm{time}_M : \Sigma^* \to \mathbb{N}$ is defined to return on input $\mathbf{a}$ the number of steps that it takes until the machine terminates on input $\mathbf{a}$. A running time of the machine is then defined to be a function $t : \mathbb{N} \to \mathbb{N}$ such that

$$\forall \mathbf{a} \in \Sigma^* : \mathrm{time}_M(\mathbf{a}) \leq t(|\mathbf{a}|). \tag{rt}$$

For an oracle Turing machine, each of the time functions $\mathrm{time}_{M^\varphi}$ may be different. Thus, the above definition has to be replaced. The most common replacement is to replace t by a higher type object as discussed in the previous section. However, there exist other approaches of how to replace this definition in literature that stay with functions of type $\mathbb{N} \to \mathbb{N}$ for

running times of oracle machines. To distinguish these objects from the time function and the running times from second-order complexity theory, we refer to the bounds as 'step-counts' instead of 'running times'. One example of a definition in this vein has been investigated by Stephen Cook [2]. He bounds the steps an oracle Turing machine may take by modifying (rt) as follows: He replaces $|\mathbf{a}|$ by the maximum m_{M^φ} of $|\mathbf{a}|$ and the biggest length of any return value of the oracle in the run of M^φ. Then he additionally universally quantifies over $\varphi \in \mathcal{B}$. Thus ending up with

$$\forall \varphi \in \mathcal{B}, \forall \mathbf{a} \in \Sigma^* \colon \operatorname{time}_{M^\varphi}(\mathbf{a}) \leq t(m_{M^\varphi}). \tag{ot}$$

He refers to the class of functionals that can be computed by an oracle machine such that the above is fulfilled for some polynomial OPT (for 'oracle polynomial time').

3.1 Step-counts

We use a more complicated definition that turns out to be considerably more well-behaved.

▶ **Definition 16.** Let $M^?$ be an oracle Turing machine. For a given oracle φ and a given input $\mathbf{a}$ denote the content of the oracle answer tape in the k-th step of the computation by $\mathbf{b}_k$. Define the **length revision function** $o_{\varphi,\mathbf{a}} : \mathbb{N} \to \mathbb{N}$ recursively as follows:

$$o_{\varphi,\mathbf{a}}(0) := |\mathbf{a}| \quad \text{and} \quad o_{\varphi,\mathbf{a}}(n+1) := \max\{o_{\varphi,\mathbf{a}}(n), |\mathbf{b}_{n+1}|\}.$$

Note that $o_{\varphi,\mathbf{a}}(k+1) > o_{\varphi,\mathbf{a}}(k)$ means that in the k-th step of the computation, the machine asked an oracle query and the answer was bigger than both the input $\mathbf{a}$ and any of the answers the oracle has given earlier in the computation. We call this a **length revision** as it means that it became apparent to the machine that its input (the oracle) is bigger than what the previous evidence indicated.

For an oracle machine $M^?$ with a fixed oracle $\varphi \in \mathcal{B}$ let $\operatorname{time}_{M^\varphi}(\mathbf{a}) \in \mathbb{N} \cup \{\infty\}$ be the number of steps that the computation of M^φ takes on input $\mathbf{a}$. I.e. the machine is explicitly allowed to diverge on some inputs.

▶ **Definition 17** (compare Figure 1)**.** A function $t \colon \mathbb{N} \to \mathbb{N}$ is a **step-count** for an oracle Turing machine $M^?$ if

$$\forall \varphi \in \mathcal{B}, \forall \mathbf{a} \in \Sigma^*, \forall n \leq \operatorname{time}_{M^\varphi}(\mathbf{a}) \colon n \leq t(o_{\varphi,\mathbf{a}}(n)). \tag{sc}$$

Denote the set of all functionals on the Baire space that can be computed by an oracle Turing machine that has a polynomial step-count by PSC.

Note that in contrast to equations (rt) and (ot), the above is not void if the machine diverges on some inputs. The relationship between termination of a machine and the existence of a step-count is quite involved. For instance: If a machine has a step-count and diverges, then the machine queries the oracle an infinite number of times. Furthermore, if there is an integer bound on the length of all return values of an oracle, then every machine that has a step-count terminates when given that oracle and an arbitrary input.

Note that $o_{\varphi,\mathbf{a}}(\operatorname{time}_{M^\varphi}(\mathbf{a}))$ is by definition the maximum of the length of $\mathbf{a}$ and the biggest oracle query done in the computation of $M^\varphi(\mathbf{a})$. This number was previously called m_{M^φ}. Thus, Stephen Cook's class OPT (see equation (ot)) can be reproduced by not quantifying over all $n \leq \operatorname{time}_{M^\varphi}(\mathbf{a})$ but only considering the case $n = \operatorname{time}_{M^\varphi}(\mathbf{a})$. In upcoming proofs it is used that it is possible to clock a machine while basically maintaining the same step-count. This is done by checking in each step that the requirement of being a step-count is fulfilled.

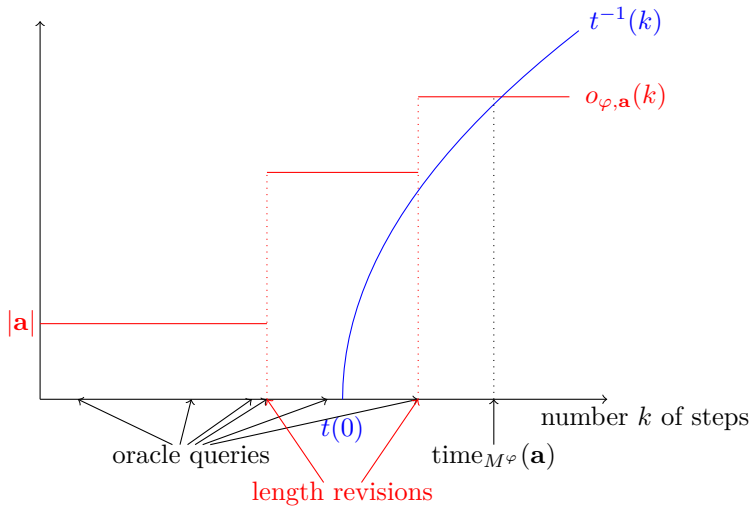


Figure 1 Verifying that φ and $\mathbf{a}$ are not a counterexample of t being a step-count. Under the assumption that t is invertible on the set $[t(0), \infty)$.

Note that this is a priory not possible for the machines used by Cook without increasing the step-count considerably. His definition allows to retroactively justify high time-consumption early in the computation by a big oracle answer late in the computation.

The very example that Cook used to disregard the class OPT as a candidate for the class of polynomial-time functionals can be used to also disregard the class of total functionals that are computed by a machine that allows a polynomial step-count:

▶ **Example 18** (PSC $\not\subseteq$ P)**.** The total functional $F : \mathcal{B} \to \mathcal{B}$ defined by

$$F(\varphi)(\mathbf{a}) := \varphi^{|\mathbf{a}|}(0)$$

can be computed by an oracle Turing machine that has a polynomial step-count but does not carry polynomial-time computable input to polynomial-time computable output.

To see that this machine has a polynomial step-count, note that it can be computed by the machine that proceeds as follows: It copies the input to the memory tape and writes 0 to the oracle query tape. Then as long as the memory tape is not empty it repeats the following steps: First copies the content of the oracle answer band to the oracle query band. Then it removes the content of the last non-empty cell from the memory band. Finally it enters the oracle query state. When the memory tape is empty it copies the content of the oracle answer band to the output tape and enters the termination state.

Copying a string of length n takes $\mathcal{O}(n)$ steps. The length of the string that has to be copied is always bounded by the previous oracle answers. The loop is carried out exactly $|\mathbf{a}|$ times. Therefore, there is some step-count from $\mathcal{O}(n^2)$.

To verify that the functional does not preserve the class of polynomial-time computable functionals consider the polynomial-time computable functional $\psi(\mathbf{a}) := \mathbf{aa}$. Note that

$$F(\psi)(\mathbf{a}) = \psi^{|\mathbf{a}|}(0) = 0^{2^{|\mathbf{a}|}}.$$

Therefore, writing $F(\psi)(\mathbf{a})$ takes at least $2^{|\mathbf{a}|}$ steps and thus $F(\psi)$ cannot be polynomial-time computable.

This means that further restrictions are necessary. In [2] this is the point where Stephen Cook decides to use polynomial-time computable functionals. This paper presents a different set of restrictions that can be used.

3.2 Finite length-revision

Let $M^?$ be an oracle Turing machine that always terminates. Then for any oracle φ and any string $\mathbf{a}$ the computation of M^φ on $\mathbf{a}$ is finite and only queries the oracle a finite number of times. Note that the number $\#o_{\varphi,\mathbf{a}}(\mathbb{N})$ of elements of the image of the the length revision function coincides with the number of length revisions that happens during the computation on oracle φ and input $\mathbf{a}$. Thus, the following statement holds true:

$$\forall \varphi \in \mathcal{B}, \forall \mathbf{a} \in \Sigma^* \colon \exists N \in \mathbb{N} \colon \#o_{\varphi,\mathbf{a}}(\mathbb{N}) \leq N.$$

In general N depends on the choice of the oracle and the string. Our restriction on the behavior of the machine is that there is an N that works independently of the choice of the oracle and the input.

▶ **Definition 19.** We say that an oracle Turing machine $M^?$ has **finite length-revision** if there is an integer N such that no matter what the oracle and the input are, no more than N length revisions happen. That is, if its length revision functions $o_{\varphi,\mathbf{a}}$ fulfill

$$\exists N \in \mathbb{N} \colon \forall \varphi \in \mathcal{B}, \forall \mathbf{a} \in \Sigma^* \colon \#o_{\varphi,\mathbf{a}}(\mathbb{N}) \leq N. \tag{flr}$$

We denote the set of all functionals on the Baire space that can be computed by machines with finite length-revision by FLR.

Finite length revision does a priori neither restrict the number of oracle questions nor the length of the oracle answers: The restriction is that there is a finite number of length revisions, that is, only a finite number of times it happens that a query is asked such that the answer is strictly bigger than the input and any earlier oracle answer.

▶ **Example 20** (P $\not\subseteq$ FLR)**.** Consider the functional

$$F : \mathcal{B} \to \mathcal{B}, \quad F(\varphi)(\mathbf{a}) := 0^{\max\{|\varphi(0^n)| \mid n \leq |\mathbf{a}|\}}.$$

The straightforward implementation asks n queries, compares their lengths and returns the maximum. This can be done in time $P(l,n) = C(n + n \cdot l(n)) + C$ for some $C \in \mathbb{N}$. However, since $|\varphi(0^n)|$ may be strictly increasing when n increases, this machine does not have finite length revision.

Indeed, no machine with finite length revision can compute F, as can be proven via contradiction as follows: Assume that there was such a machine $M^?$. Let N be a bound on the length-revisions $M^?$ does. Define an oracle such that the output of $M^\varphi(0^{N+1})$ is incorrect as follows: Let $\mathbf{a}_1$ be the first oracle query that is asked in the run of the machine $M^\varphi(0^N)$. Set $\varphi(\mathbf{a}_1) := 0^{N+1}$. Thus, a length-revision happens. Let $\mathbf{a}_2$ be the next oracle query that the machine poses. Set $\varphi(\mathbf{a}_2) := 0^{N+2}$. This means that another length revision happens. Carry on in that way until $\varphi(a_N)$ is set to 0^{2N}. After asking the query $\mathbf{a}_N$, the machine can not ask another query as we may as well set the return value to be bigger again and no further length revision is allowed.

Note that the run of the machine on 0^N is identical for any oracle that fulfills $\psi(\mathbf{a}_i) = 0^{N+i}$. Let M be the number of steps the machine M^ψ takes for any of these oracles to terminate. There are $N+1$ strings of the form 0^n for $n \leq N$. Thus, at least one of these strings is not

contained within $\mathbf{a}_1, \ldots, \mathbf{a}_N$. Let 0^m be this string. Let φ be the string function defined as follows:

$$\varphi(\mathbf{b}) = \begin{cases} 0^{N+i} & \text{if } \mathbf{b} = \mathbf{a}_i \\ 0^{M+1} & \text{if } \mathbf{b} = 0^m \\ \varepsilon & \text{otherwise.} \end{cases}$$

Obviously, the run of M^φ on 0^N coincides with the one described above. Therefore the return value can have at most M bits. Since $m \leq N$ it holds that $\left|F(\varphi)(0^N)\right| \geq |\varphi(0^m)| \geq M + 1$. Thus M^φ can on input 0^N not produce the right return value.

3.3 Strong polynomial-time computability

While neither finite length revision nor having a step-count implies termination of the machine, the combination does: We mentioned that a machine that has a step-count may only diverge with oracle φ if there is no bound on the oracle answers. This, however, is forbidden by finite length revision. Therefore, if $M^?$ is a machine that has finite length-revision and a step-count, then the computation of $M^?$ with any oracle and on any input terminates.

▶ **Definition 21.** Call a functional $F : \mathcal{B} \to \mathcal{B}$ **strongly polynomial-time computable** if there is an oracle Turing machine computing F that has both finite length-revision and a polynomial step-count (see Definition 17). We denote the set of all strongly polynomial-time computable operators by SP

As the name suggests, strong polynomial-time computability implies polynomial-time computability.

▶ **Lemma 22** (SP $\subseteq$ P)**.** *Any total strongly polynomial-time computable functional is polynomial-time computable.*

Proof. Let $M^?$ be the Turing machine that verifies that the total functional is strongly polynomial time computable, p a polynomial step-count of the machine and N a bound of the number of length revisions it does. To see that the machine runs in polynomial time fix some arbitrary oracle φ and a string $\mathbf{a}$. By the definition of being a step-count, the first oracle query in the run of M^φ on input $\mathbf{a}$ that leads to a length revision is done after at most $p(|\mathbf{a}|)$ steps and has at most this number of bits. Thus the return value of the oracle has at most length $|\varphi|(p(|\mathbf{a}|))$. Therefore, again since p is a step-count, the next oracle query that leads to a length revision can not have more than $p(|\varphi|(p(|\mathbf{a}|)))$ bits. Repeating the above argument N times and using that N is a bound of the number of length revisions proves that the computation terminates within at most $(p \circ |\varphi|)^N(p(|\mathbf{a}|))$ steps. That is, that the second order polynomial $P(l, n) := (p \circ l)^N(p(n))$ is a running time of $M^?$. ◀

Strong polynomial-time is strictly more restrictive than polynomial-time computability.

▶ **Lemma 23** (SP $\subsetneq$ P)**.** *There exists a polynomial-time computable functional that is not computable with finite length-revision. In particular, this functional is not strongly polynomial-time computable.*

Proof. An functional polynomial-time computable functional that can not be computed by a machine with finite length revision was discussed in detail in Example 20. Since SP $\subseteq$ FLR, this proves that the inclusion SP $\subseteq$ P from the previous result is strict. ◀

A candidate for a natural example of an operator from analysis that is not strongly polynomial-time computable is constructed in [1].

3.4 Compatibility with relativization

For strong polynomial-time computability, relativized notions can be introduced analogously to Section 2.1: Let $A \subseteq \mathcal{B}$. A machine $M^?$ is said to run in A-restricted strongly polynomial time if the number of length revisions $M^?$ does on oracles from A is bounded by a number and there is a polynomial step-count that is valid whenever the oracle is from A. That is if the formulas (flr) from Definition 17 and (sc) from Definition 19 are fulfilled if '$\forall \varphi \in \mathcal{B}$' is replaced by '$\forall \varphi \in A$'. Again, we denote the set of all functionals whose domain is A and that can be computed by an A-restricted strongly polynomial-time machine by $\mathrm{SP}(A)$ and the set of all functionals whose domain is contained in A and that have a total strongly polynomial-time computable extension by $\mathrm{SP}|_A$. For strong polynomial-time computability these classes coincide. This may be interpreted as strong polynomial-time computability being more well behaved with respect to partial functionals.

▶ **Lemma 24** ($\mathrm{SP}(A) = \mathrm{SP}|_A$)**.** *Any A-restricted strongly polynomial-time computable functional has a total strongly polynomial-time computable extension.*

Proof. Let $F : A \to \mathcal{B}$ be an A-restricted strongly polynomial-time computable functional and let $M^?$ be a machine that witnesses the strong polynomial-time computability of the functional. Let N be maximum number of length revisions $M^?$ does on any oracle from A and let p be a polynomial step-count valid for input from A. Define a new machine $\tilde{M}^?$ as follows: $\tilde{M}^?$ starts by initializing a counter with N written on it. Furthermore it saves the length of the input string and produces the coefficients of p on the memory tape. It applies the polynomial p to the length of the input and initializes a second counter holding this value. Now it follows the exact same steps $M^?$ does as long as no oracle query is done and meanwhile counts down the second counter. If the second counter hits zero, it terminates and returns ε. Whenever an oracle call is done, the machine checks whether the answer is longer than the answers seen before, if so, it decreases the first counter. If the counter was already zero, it terminates and returns ε. If it was not, it writes the maximum of the previous content and polynomial p applied to the length of the return value minus the number of steps taken so far to where it originally noted the length of the input. It applies the polynomial to this new value and ads the difference to the previous value to the second counter. Then it continues as before.

It is clear that the machine described above does at most $N + 1$ length revision, that it has a polynomial step-count (that depends only on p and N) and that whenever the oracle is from A, none of the counters will hit zero and $\tilde{M}^\varphi$ and M^φ produce the same values in the end. Thus $\tilde{M}^?$ computes a total strongly polynomial-time computable extension of F. ◀

3.5 Comparison to polynomial-time on Σ^{**}

Recall that originally polynomial-time computability was only defined for machines that compute total functions. Kawamura and Cook's framework for complexity of operators in analysis, however, does not require a realizer to have a total polynomial-time computable extension, but instead gives a new definition of what polynomial-time computability of a functional on Σ^{**} means. Earlier, this notion of complexity was called being Σ^{**}-restricted polynomial-time computable and the class of these functionals was denoted by $P(\Sigma^{**})$. This section proves that, for a functional whose domain is contained in Σ^{**}, the four notions of having a total extension from P or SP, or having an extension to all of Σ^{**} from $\mathrm{SP}(\Sigma^{**})$ or $P(\Sigma^{**})$ coincide. Part of this was already proven in Lemma 24, which implies that having an extension from SP and from $\mathrm{SP}(\Sigma^{**})$ are equivalent. This implies that the domain of the functional considered in Example 20 was necessarily not contained in Σ^{**}.

▶ **Theorem 25** ($\mathrm{SP}(\Sigma^{**}) = \mathrm{P}(\Sigma^{**})$)**.** *A functional is Σ^{**}-restricted polynomial-time computable if and only if it is strongly polynomial-time computable.*

Proof. That $\mathrm{SP}(\Sigma^{**}) \subseteq \mathrm{P}(\Sigma^{**})$ follows from previous results: By Lemma 24 every element of $\mathrm{SP}(\Sigma^{**})$ has a total, strongly polynomial-time computable extension. By Lemma 22, this extension is polynomial-time computable and therefore in particular Σ^{**}-restricted polynomial-time computable.

For the other direction let $M^?$ be a machine that computes some $F : \Sigma^{**} \to \mathcal{B}$ in Σ^{**}-restricted polynomial time. Define a new machine $\tilde{M}^?$ as follows: When given φ as oracle and a string $\mathbf{a}$ as input, the machine computes $P(l, m)$ with $m := |\mathbf{a}|$ and $l(n) := |\varphi(0^n)|$. This can be done by carrying out a finite sequence of oracle queries and applications of (possibly multivariate) polynomials. This sequence of operations (in particular what polynomials are applied) is determined solely from the second-order polynomial P. (More information about how to construct the polynomials can be found in Appendix A.) It writes the result into a counter, does a final query of the oracle of this length and then caries out the computations $M^?$ does on oracle φ and input $\mathbf{a}$ while counting this counter down. If the counter runs empty or an oracle answer bigger than any previous answer is encountered it terminates and returns ε. If $M^?$ terminates before this happens, it returns $M^{\varphi}(\mathbf{a})$.

While computing the value of the second-order polynomial, the number of oracle calls machine $\tilde{M}^?$ does, and therefore also the number of length revisions, is bounded by the number of applications of the length function done in the second-order polynomial P. The computation is aborted if the simulation of the machine $M^?$ leads to another length revision, thus the machine $\tilde{M}$ has finite length revision. To see that the function has a polynomial step-count note that applying polynomials from a fixed finite set of polynomials is unproblematic due to the time-constructibility of polynomials and that the simulation takes at most a number of steps that is linear in the size of the second to last oracle query before the simulation started. Thus, the machine $\tilde{M}^?$ runs in strongly polynomial time and therefore also in Σ^{**}-restricted strongly polynomial time.

Finally argue that the machine $\tilde{M}^?$ computes an extension of F: Whenever φ is length-monotone, the value $P(l, m)$ written to the counter coincides with $P(|\varphi|, |\mathbf{a}|)$. Since $M^?$ computes F in Σ^{**}-restricted time bounded by P it does not happen that the timer runs out. Due to the final oracle query $\tilde{M}^?$ does before starting to simulate $M^?$ and the length monotonicity of φ it does not happen that the simulation of $M^?$ triggers another length revision. Thus, the machine $\tilde{M}^?$ returns $M^{\varphi}(\mathbf{a})$ and therefore computes a total extension of F. ◀

4 Conclusion

The results of this paper are tightly connected to questions of whether or not it is possible to add clocks to certain machines. Clocking is a standard procedure to increase the domain of machines while maintaining its behavior on a set of 'important' oracles and inputs. For regular Turing machines, clocking allows to turn any machine that runs in polynomial time on the inputs the user cares about into a machine that actually runs in polynomial time: Take the polynomial that bounds the running time on the important inputs and in each step check if this number of steps was exceeded. This machine runs in about the same time as the original machine due to the time constructibility of polynomials. When moving to oracle Turing machines, the polynomials have to be replaced by second-order polynomials and unfortunately, these turn out not to be time constructible. Thus, for oracle Turing machines the above procedure does not extend in a straight forward manner. Indeed, Theorem 11 proves that it is impossible to clock a polynomial-time machine in general.

The framework of Kawamura and Cook and their restriction to length-monotone functions avoids the implications of this by imposing assumptions on the domains of the functionals that are considered. The notion of strong polynomial-time computability introduced in this paper tackles the same problem from another angle: For any domain it introduces a class of functionals such that clocking is possible and thus total polynomial-time computable extensions exist. Furthermore, the extensions can be chosen strongly polynomial-time computable again. However, for total functions, strong polynomial-time computability is a strictly stronger condition than polynomial-time computability.

Strong polynomial-time computability is another step towards the expectations of programmers what programs with subroutine calls should be fast. It removes dependencies of the running time on information that can not be read from the oracle in a fast way. Strong polynomial-time computability also has an intuitive meaning: A program with a subroutine that runs with length revision number N and polynomial step-count can be turned into another program with subroutine calls that computes the same output in about the same time but additionally behaves as follows: Whenever you start the program, it provides you with a time promise. Not a promise that it will terminate in the specified time, but instead a promise that within this time it will beep, revise the time promise and provide you with a reason for the revision in form of a complicated return value of a function call. Furthermore the machine will at most beep N times.

The research presented in this paper provides several starting-points for further investigations. For instance, the notion of a step-count can easily be tweaked to count the number of memory cells that are in use instead of the steps the computation takes. Such a notion could for instance be compared to the classes of operators computable with restricted space that have been introduced to the Framework of Kawamura and Cook by Kawamura and Ota [9]. This seems in particular promising as it turns out that space restricted computation in the presence of oracles is more complicated in the framework of Kawamura and Cook: To preserve the usual inclusions of complexity classes the model of computation has to be tweaked. The regular oracle Turing machines need to be replaced by machines that have a finite height stack of oracle tapes. The height of the oracle stack is another characteristic number of such a machine. Availability of an alternative point of view on space restricted computation might lead to a better understanding. Other possible follow-ups would be to investigate non-deterministic or probabilistic computation.

Acknowledgements. The authors thank an anonymous referee for the extensive feedback that lead to many improvements of the paper. The second author thanks Matthias Schröder for extended discussion on the topics of the paper.

References

1 Franz Brauße and Florian Steinberg. A minimal representation for continuous functions. `https://arxiv.org/abs/1703.10044`, 2017. Preprint.

2 Stephen A. Cook. Computational complexity of higher type functions. In *Proceedings of the International Congress of Mathematicians, Vol. I, II (Kyoto, 1990)*, pages 55–69. Math. Soc. Japan, Tokyo, 1991.

3 Hugo Férée, Walid Gomaa, and Mathieu Hoyrup. Analytical properties of resource-bounded real functionals. *J. Complexity*, 30(5):647–671, 2014. `doi:10.1016/j.jco.2014.02.008`.

4 Hugo Férée and Mathieu Hoyrup. Higher order complexity in analysis, 2013. CCA. URL: `https://hal.inria.fr/hal-00915973/document`.

5 Hugo Férée and Martin Ziegler. On the computational complexity of positive linear functionals on c[0;1], 2015. MACIS conference. URL: https://hugo.feree.fr/macis2015.pdf.

6 B. M. Kapron and S. A. Cook. A new characterization of type-2 feasibility. *SIAM J. Comput.*, 25(1):117–132, 1996. doi:10.1137/S0097539794263452.

7 Akitoshi Kawamura. *Computational Complexity in Analysis and Geometry.* PhD thesis, University of Toronto, 2011.

8 Akitoshi Kawamura and Stephen Cook. Complexity theory for operators in analysis. *ACM Trans. Comput. Theory*, 4(2):5:1–5:24, May 2012. doi:10.1145/2189778.2189780.

9 Akitoshi Kawamura and Hiroyuki Ota. Small complexity classes for computable analysis. In *Mathematical foundations of computer science 2014. Part II*, volume 8635 of *Lecture Notes in Comput. Sci.*, pages 432–444. Springer, Heidelberg, 2014. doi:10.1007/978-3-662-44465-8_37.

10 Akitoshi Kawamura and Arno Pauly. Function spaces for second-order polynomial time. In *Language, life, limits*, volume 8493 of *Lecture Notes in Comput. Sci.*, pages 245–254. Springer, Cham, 2014. doi:10.1007/978-3-319-08019-2_25.

11 Akitoshi Kawamura, Florian Steinberg, and Martin Ziegler. Complexity theory of (functions on) compact metric spaces. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS'16, pages 837–846, New York, NY, USA, 2016. ACM. doi:10.1145/2933575.2935311.

12 Akitoshi Kawamura, Florian Steinberg, and Martin Ziegler. Towards computational complexity theory on advanced function spaces in analysis. In Arnold Beckmann, Laurent Bienvenu, and Nataša Jonoska, editors, *Pursuit of the Universal: 12th Conference on Computability in Europe, CiE 2016, Paris, France, June 27 – July 1, 2016, Proceedings*, pages 142–152. Springer, Cham, 2016. doi:10.1007/978-3-319-40189-8_15.

13 Aktioshi Kawamura and Florian Steinberg. Polynomial running times for polynomial-time oracle machines. https://arxiv.org/abs/1704.01405, 2017. Preprint.

14 Branimir Lambov. The basic feasible functionals in computable analysis. *J. Complexity*, 22(6):909–917, 2006. doi:10.1016/j.jco.2006.06.005.

15 Kurt Mehlhorn. Polynomial and abstract subrecursive classes. *J. Comput. System Sci.*, 12(2):147–178, 1976. Sixth Annual ACM Symposium on the Theory of Computing (Seattle, Wash., 1974).

16 Matthias Schröder and Florian Steinberg. Bounded time computation on metric spaces and Banach spaces. https://arxiv.org/abs/1701.02274, 2017. Preprint; extended abstract accepted for LICS 2017 conference.

A Descriptions of second-order polynomials

This appendix closes two gaps in the proofs of this paper: In the proof of Theorem 25 it was claimed that evaluating a second-order polynomial can be done by 'carrying out a finite sequence of oracle queries and applications of (possibly multivariate) polynomials' whenever queries are know that force maximal length of the return value. This appendix proves that there is such a collection of multivariate polynomials for each second-order polynomial. As a side product it provides a more elegant proof of Lemma 3 than the suggested tedious induction of the term structure.

Recall from Section 2.1 that a second-order polynomial is a function $P : \mathbb{N}^{\mathbb{N}} \times \mathbb{N} \to \mathbb{N}$ whose values are determined from a finite list of applications of certain rules. These rules can be understood as introduction of first-order polynomials, closure under addition, closure under multiplication and closure under function application. It seems reasonable to bundle the uses of the 'closure under addition' and the 'closure under multiplication' rules that

happen between two uses of the 'closure under function application' rule together to applying a multivariate polynomial. Formally this procedure can be described as follows:

▶ **Definition 26.** A **polynomial tree** is a finite tree T whose nodes are elements of $\mathbb{N}[X_0, \dots, X_k]$ where k coincides with the number of children the node has and there is a specified linear order on the children of each node.

Given a polynomial tree, recursively assign to each node a second-order polynomial: To a leaf t assign the second order polynomial $(l, n) \mapsto t(n)$. Now assume that second-order polynomials $P_1, \dots, P_k$ were assigned to each of the children $t_1, \dots, t_k$ of a node t. Assign to t the second-order polynomial

$$(l, n) \mapsto t(n, l(P_1(l, n)), \dots, l(P_k(l, n))) = t(n, P_1^+, \dots, P_k^+).$$

▶ **Definition 27.** A polynomial tree is called a **description** of a second-order polynomial P if P is assigned to the root of the tree by the above procedure.

Note that there may exist many different descriptions of the same second-order polynomial. For instance, both of the polynomial trees below are descriptions of the second-order polynomial $(l, n) \mapsto 2l(n)$.

Whether or not such ambiguities can completely be avoided seems to be related to whether or not the operation $P \mapsto P^+$ is injective. It is not known to the authors whether or not injectivity of this mapping holds and it is not relevant to the content of this paper. However, removing the ambiguities in descriptions would provide a normal form theorem for second-order polynomials which seems highly desirable.

▶ **Lemma 28.** *Every second-order polynomial has a description.*

Proof. For the base case note that the a description consisting of a single node $p \in \mathbb{N}[X_0]$ is a description of the second-order polynomial $(l, n) \mapsto p(n)$.

To obtain a description of the point-wise sum $P + Q$ from descriptions of P and of Q, let $t_P \in \mathbb{N}[X_0, \dots, X_k]$ be the polynomial at the root of Ps description and $t_Q \in \mathbb{N}[X_0, \dots, X_m]$ the polynomial at the root of Qs description. A description of $P + Q$ is given by merging the root of the two descriptions to a node labeled with the polynomial

$$\tilde{t}(X_0, \dots, X_{k+m+1}) := t_P(X_0, \dots, X_k) + t_Q(X_{k+1}, \dots, X_{k+m+1}).$$

For the point-wise product replace $t_P + t_Q$ in the above procedure by $t_P \cdot t_Q$.

Finally note that if P is a second-order polynomial and T a description of P then adding a single node containing the polynomial X_1 above the root of T is a description of P^+. ◀

This completes the proof of Theorem 25: Pick a description of the second-order polynomial P that is a running time of the operator. This second-order polynomial can be evaluated by going through the description, applying a regular polynomial to the length of the input for each leaf of the description, doing an oracle query for each edge of the description and evaluating a multivariate polynomial for each node of the description that has children.

Descriptions are very convenient to reason about second-order polynomials. For instance a direct proof of Lemma 3 can be given. Recall that this lemma stated that whenever P and Q are second-order polynomials, then so are the mappings

$$(l, n) \mapsto P(Q(l, \cdot), n) \quad \text{and} \quad (l, n) \mapsto P(l, Q(l, n)).$$

Proof of Lemma 3. A description of the latter can be specified by replacing each leaf p of a description of P with a description of Q where the root t of the description of Q is replaced by $p \circ t$. For the former one each edge of in a description of P has to be replaced with a description of Q (where a copy of the part of the description of P below the edge is appended to each leaf of the description of Q and there are compositions again in the roots and the leafs). ◀

Types as Resources for Classical Natural Deduction

Delia Kesner[1] and Pierre Vial[2]

1 **IRIF (CNRS and Université Paris-Diderot), Paris, France**
2 **IRIF (CNRS and Université Paris-Diderot), Paris, France**

Abstract

We define two resource aware typing systems for the λ_μ-calculus based on non-idempotent *intersection* and *union* types. The non-idempotent approach provides very simple combinatorial arguments – based on decreasing measures of type derivations – to characterize head and strongly normalizing terms. Moreover, typability provides upper bounds for the length of head-reduction sequences and maximal reduction sequences.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases lambda-mu-calculus, classical logic, intersection types, normalization

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.24

1 Introduction

A few years after Griffin [22] observed that Feilleisen's C operator can be typed with the double-negation elimination, Parigot [32] made a major step in extending the Curry-Howard from intuitionistic to classical logic by proposing the λ_μ-calculus as a simple term notation for classical natural deduction proofs. Other calculi were proposed since then, as for example Curien-Herbelin's $\lambda\mu\tilde{\mu}$-calculus [11] based on classical sequent calculus.

Simple types are known to be unable to type some normalizing term, for instance the normal form $\Delta = \lambda x.xx$. *Intersection* types, pioneered by Coppo and Dezani [9, 10], extend simple types by resorting to a new constructor $\cap$ for types, allowing the assignment of a type of the form $((\sigma \Rightarrow \sigma) \cap \sigma) \Rightarrow \sigma$ to the term Δ. The intuition behind a term t of type $\tau_1 \cap \tau_2$ is that t has both types τ_1 and τ_2. The intersection operator $\cap$ is to be understood as *idempotent* ($\sigma \cap \sigma = \sigma$), *commutative* ($\sigma \cap \tau = \tau \cap \sigma$), and *associative* ($(\sigma \cap \tau) \cap \delta = \sigma \cap (\tau \cap \delta)$) laws. Among other applications, intersection types have been used as a *behavioural* tool to reason about several operational and semantical properties of programming languages. For example, a λ-term/program t is strongly normalizing/terminating if and only if t can be assigned a type in an appropriate intersection type assignment system.

This technology turns out to be a powerful tool to reason about *qualitative* properties of programs, but not about *quantitative* ones. Indeed, *e.g.* there is a type system assigning a type to a term t if and only if t is head normalizing, but the type derivations give no information about the number of head-reduction steps needed to head-normalize t, because of idempotency. In constrast, after the pioneering works of Gardner [19] and Kfoury [27], D. de Carvalho [14, 15] established a relation between the size of a typing derivation in a non-idempotent intersection type system for the lambda-calculus and the head/weak-normalization execution time of head/weak-normalizing lambda-terms, respectively. Non-idempotent types have recently received a lot of attention in the domain of semantics of programming languages from a quantitative perspective (see for example [6]), notably because they are closely related

to Girard's translation of intuitionistic logic into linear logic (according to which $A \Rightarrow B$ becomes $!A \multimap B$).

The case of the λ_μ-calculus: The non-idempotent intersection and union types for lambda-mu-calculus that we present in this article can be seen as a quantitative refinement of Girard's translation of classical logic into linear logic. Different qualitative and/or quantitative models for classical calculi were proposed in [34, 37, 39, 3], thus limiting the characterization of operational properties to head-normalization. Intersection and union types were also studied in the framework of classical logic [30, 36, 28, 17], but no work adresses the problem from a quantitative perspective. Type-theoretical characterization of strong-normalization for classical calculi were provided both for λ_μ [38] and $\lambda\mu\tilde{\mu}$-calculus [17], but the (idempotent) typing systems do not allow to construct decreasing measures for reduction, thus a resource aware semantics cannot be extracted from those interpretations. Combinatorial strong normalization proofs for the λ_μ-calculus were proposed for example in [12], but they do not provide any explicit decreasing measure, and their use of structural induction on simple types does not work anymore with intersection types, which are more powerful than simple types as they do not only ensure termination but also characterize it. Different small step semantics for classical calculi were developed in the framework of neededness [4, 33], without resorting to any resource aware semantical argument.

In this paper we define a resource aware type system for the λ_μ-calculus based on non-idempotent *intersection* and *union* types. The non-idempotent approach provides very simple combinatorial arguments, only based on a decreasing *measure*, to characterize head and strongly normalizing terms by means of typability. In the well-known case of the λ-calculus, the measure $\mathtt{sz}(\Pi)$ of a derivation Π is simply given by the number of its nodes. This approach cannot be straightforwardly adapted to λ_μ, and we need now to take into account the structure (*multiplicity* and *size*) of certain types appearing in the types derivations.

By lack of space we cannot provide in this submission all the proofs of our results, but we refer the interested reader to the extended detailed version available at [26].

2 The λ_μ-Calculus

This section gives the syntax (Sec. 2.1) and the operational semantics (Sec. 2.2) of the $\lambda_\mu-$calculus [32]. But before this we first introduce some preliminary general notions of rewriting that will be used all along the paper, and that are applicable to any system $\mathcal{R}$. We denote by $\to_\mathcal{R}$ the (one-step) reduction relation associated to system $\mathcal{R}$. We write $\to_\mathcal{R}^*$ for the reflexive-transitive closure of $\to_\mathcal{R}$, and $\to_\mathcal{R}^n$ for the composition of n-steps of $\to_\mathcal{R}$, thus $t \to_\mathcal{R}^n u$ denotes a finite $\mathcal{R}$-reduction sequence of length n from t to u. A term t is in $\mathcal{R}$-normal form, written $t \in \mathcal{R}$-nf, if there is no t' s.t. $t \to_\mathcal{R} t'$; and t has an $\mathcal{R}$-normal form iff there is $t' \in \mathcal{R}$-nf such that $t \to_\mathcal{R}^* t'$. A term t is said to be strongly $\mathcal{R}$-normalizing, written $t \in \mathcal{SN}(\mathcal{R})$, iff there is no infinite $\mathcal{R}$-sequence starting at t.

2.1 Syntax

We consider a countable infinite set of **variables** $x, y, z, \ldots$ (resp. **continuation names** $\alpha, \beta, \gamma, \ldots$). The set of **objects** ($\mathcal{O}_{\lambda_\mu}$), **terms** ($\mathcal{T}_{\lambda_\mu}$) and **commands** ($\mathcal{C}_{\lambda_\mu}$) of the λ_μ-calculus are given by the following grammars

(**objects**)	o	$::=$	$t \mid c$
(**terms**)	t, u, v	$::=$	$x \mid \lambda x.t \mid tu \mid \mu\alpha.c$
(**commands**)	c	$::=$	$[\alpha]t$

We write $\mathcal{T}_\lambda$ for the set of λ-terms. We abbreviate $(\dots((tu_1)u_2)\dots u_n)$ as $tu_1\dots u_n$ or $t\overline{u}$ when n is clear from the context. The grammar extends λ-terms with two new constructors: commands $[\alpha]t$ and μ-abstractions $\mu\alpha.c$. **Free** and **bound variables** of objects are defined as expected, in particular $\mathtt{fv}(\mu\alpha.c) := \mathtt{fv}(c)$ and $\mathtt{fv}([\alpha]t) := \mathtt{fv}(t)$. **Free names** of objects are defined as expected, in particular $\mathtt{fn}(\mu\alpha.c) := \mathtt{fn}(c) \setminus \{\alpha\}$ and $\mathtt{fn}([\alpha]t) := \mathtt{fn}(t) \cup \{\alpha\}$. **Bound names** are defined accordingly.

We work with the standard notion of **α-conversion** *i.e.* renaming of bound variables and names, thus for example $[\delta](\mu\alpha.[\alpha](\lambda x.x))z \equiv [\delta](\mu\beta.[\beta](\lambda y.y))z$. **Substitutions** are (finite) functions from variables to terms specified by $\{x_1/u_1, \dots, x_n/u_n\}$ $(n \geq 0)$. **Application** of the **substitution** σ to the object o, written $o\sigma$, may require α-conversion in order to avoid capture of free variables/names, and it is defined as expected. **Replacements** are (finite) functions from names to terms specified by $\{\alpha_1 /\!\!/ u_1, \dots, \alpha_n /\!\!/ u_n\}$ $(n \geq 0)$. Intuitively, the operation $\{\alpha /\!\!/ u\}$ passes the term u as an argument to any command of the form $[\alpha]t$. Formally, the **application** of the **replacement** Σ to the object o, written $o\Sigma$, may require α-conversion in order to avoid the capture of free variables/names, and is defined as:

$$\begin{array}{rcl@{\qquad}rcl} x\{\alpha /\!\!/ u\} & := & x & (\lambda z.t)\{\alpha /\!\!/ u\} & := & \lambda z.t\{\alpha /\!\!/ u\} \\ ([\alpha]t)\{\alpha /\!\!/ u\} & := & [\alpha](t\{\alpha /\!\!/ u\})u & (tv)\{\alpha /\!\!/ u\} & := & t\{\alpha /\!\!/ u\}v\{\alpha /\!\!/ u\} \\ ([\gamma]t)\{\alpha /\!\!/ u\} & := & [\gamma]t\{\alpha /\!\!/ u\} & (\mu\gamma.c)\{\alpha /\!\!/ u\} & := & \mu\gamma.c\{\alpha /\!\!/ u\} \end{array}$$

For example, if $\mathtt{I} = \lambda z.z$, then $(x(\mu\alpha[\alpha]y)(\lambda z.zx))\{x/\mathtt{I}\} = \mathtt{I}(\mu\alpha[\alpha]y)(\lambda z.z\mathtt{I})$, and $[\alpha]x(\mu\beta.[\alpha]y)\{\alpha /\!\!/ \mathtt{I}\} = [\alpha](x\mu\beta.[\alpha]y\mathtt{I}))\mathtt{I}$.

2.2 Operational Semantics

The $\lambda\mu$-calculus is given by the set of objects introduced in Sec. 2.1 and the **reduction relation** $\to_{\lambda_\mu}$, which is the closure by all contexts of the following rewriting rules

$$\begin{array}{lcl} (\lambda x.t)u & \mapsto_\beta & t\{x/u\} \\ (\mu\alpha.c)u & \mapsto_\mu & \mu\alpha.c\{\alpha /\!\!/ u\} \end{array}$$

defined by means of the substitution and replacement application notions given in Sec. 2.1. A **redex** is a term of the form $(\lambda x.t)u$ or $(\mu\alpha.c)u$. We write $t \to_{\lambda_\mu} t'$ (or simply $t \to t'$) to denote the closure by all contexts of the reduction relation generated by the previous set of rewriting rules.

A **head-context** is a context defined by the following grammar:

$$\begin{array}{lcl} \mathcal{HO} & ::= & \mathcal{HT} \mid \mathcal{HC} \\ \mathcal{HT} & ::= & \Box t_1 \dots t_n \ (n \geqslant 0) \mid \lambda x.\mathcal{HT} \mid \mu\alpha.\mathcal{HC} \\ \mathcal{HC} & ::= & [\alpha]\mathcal{HT} \end{array}$$

A **head-normal form** is an object of the form $\mathcal{HO}[x]$, where x is any variable replacing the constant $\Box$. Thus for example $\mu\alpha.[\beta]\lambda y.x(\lambda z.z)$ is a head-normal form. An object $o \in \mathcal{O}_{\lambda_\mu}$ is said to be **head-normalizing**, written $o \in \mathcal{HN}(\lambda_\mu)$, if $o \to^*_{\lambda_\mu} o'$, for some head-normal form o'. Remark that $o \in \mathcal{HN}(\lambda_\mu)$ does not imply $o \in \mathcal{SN}(\lambda_\mu)$ while the converse necessarily holds. We write $\mathcal{HN}(\lambda)$ and $\mathcal{SN}(\lambda)$ when t is restricted to be a λ-term and the reduction system is restricted to the β-reduction rule.

A redex r in a term of the form $t := \mathcal{HO}[r]$ is called the **head-redex** of t. The reduction step $t \to_{\lambda_\mu} t'$ contracting the head-redex of t is called **head-reduction**. The reduction sequence composing head-reduction steps until head-normal form is called the **head-strategy**.

If the head-strategy starting at o terminates, then $o \in \mathcal{HN}(\lambda_\mu)$, while the converse will be stated later (*cf.* Thm. 7).

A typical example of expressivity in the λ_μ-calculus is the control operator [22] **call-cc** $:= \lambda y.\mu\alpha.[\alpha]y(\lambda x.\mu\beta.[\alpha]x)$ which gives raise to the following reduction sequence:

$$\begin{array}{l}\textbf{call-cc}\ t\ u_1\ \dots\ u_n \to_\beta (\mu\alpha.[\alpha]t(\lambda x.\mu\beta.[\alpha]x))u_1 \dots u_n \\ \to_\mu (\mu\alpha.[\alpha]t(\lambda x.\mu\beta.[\alpha]xu_1)u_1)u_2 \dots u_n \to_\mu^* \mu\alpha.[\alpha]t(\lambda x.\mu\beta.[\alpha]xu_1 \dots u_n)u_1 \dots u_n\end{array}$$

A reduction step $o \to o'$ is said to be **erasing** iff $o = (\lambda x.u)v$ and $x \notin \mathtt{fv}(u)$, or $o = (\mu\alpha.c)u$ and $\alpha \notin \mathtt{fn}(c)$. Thus *e.g.* $(\lambda x.z)y \to z$ and $(\mu\alpha.[\beta]x)\mathtt{I} \to_\mu \mu\alpha.[\beta]x$ are erasing steps. A reduction step $o \to o'$ which is not erasing is called **non-erasing**. Reduction is stable by substitution and replacement. More precisely, if $o \to o'$, then $o\{x/u\} \to o'\{x/u\}$ and $o\{\alpha /\!\!/ u\} \to o'\{\alpha /\!\!/ u\}$. These stability properties give the following corollary.

▶ **Corollary 1.** *If* $o\{x/u\} \in \mathcal{SN}(\lambda_\mu)$ *(resp.* $o\{\alpha /\!\!/ u\} \in \mathcal{SN}(\lambda_\mu)$*) , then* $o \in \mathcal{SN}(\lambda_\mu)$.

3 Quantitative Type Systems for the λ-Calculus

As mentioned before, our results rely on typability of λ_μ-terms in suitable systems with non-idempotent types. Since the λ_μ-calculus embeds the λ-calculus, we start by recalling the well-known [19, 14, 7] quantitative type systems for λ-calculus, called here $\mathcal{H}_\lambda$ and $\mathcal{S}_\lambda$. We then reformulate them, using a different syntactical formulation, resulting in the typing systems $\mathcal{H}'_\lambda$ and $\mathcal{S}'_\lambda$, that are the formalisms we adopt in Sec. 4 for λ_μ.

We start by fixing a countable set of **base types** $a, b, c \dots$, then we introduce two different categories of types specified by the following grammars:

$$\begin{array}{lccl}(\textbf{Intersection Types}) & \mathcal{I} & ::= & [\sigma_k]_{k\in K} \\ (\textbf{Types}) & \sigma, \tau & ::= & a \mid \mathcal{I} \Rightarrow \sigma\end{array}$$

An intersection type $[\sigma_k]_{k\in\{1..n\}}$ is a *multiset* that can be understood as a type $\sigma_1 \cap \dots \cap \sigma_n$, where $\cap$ is associative and commutative, but *non-idempotent*. The *non-deterministic* **choice** operation $\_^*$ is defined on intersection types as follows:

$$[\sigma_k]^*_{k\in K} := \begin{cases} [\tau] & \text{if } K = \emptyset \text{ and } \tau \text{ is any arbitrary type} \\ [\sigma_k]_{k\in K} & \text{if } K \neq \emptyset \end{cases}$$

Variable assignments (Γ) are functions from variables to intersection types. The **domain of** Γ is given by $\mathtt{dom}(\Gamma) := \{x \mid \Gamma(x) \neq [\,]\}$, where $[\,]$ is the empty intersection type. We write $x_1 : \mathcal{I}_1, \dots, x_n : \mathcal{I}_n$ for the assignment of domain $\{x_1, \dots, x_n\}$ mapping each x_i to $\mathcal{I}_i$. When $x \notin \mathtt{dom}(\Gamma)$, then $\Gamma(x)$ stands for $[\,]$. We write $\Gamma \wedge \Gamma'$ for $x \mapsto \Gamma(x) + \Gamma'(x)$, where $+$ is multiset union, and $\mathtt{dom}(\Gamma \wedge \Gamma') = \mathtt{dom}(\Gamma) \cup \mathtt{dom}(\Gamma')$. We write $\Gamma \setminus\!\!\setminus x$ for the assignment defined by $(\Gamma \setminus\!\!\setminus x)(x) = [\,]$ and $(\Gamma \setminus\!\!\setminus x)(y) = \Gamma(y)$ if $y \neq x$.

To present/discuss different typing systems, we consider the following derivability notions. A **type judgment** is a triple $\Gamma \vdash t : \sigma$, where Γ is a variable assignment, t a term and σ a type. A **(type) derivation** in system $\mathcal{X}$ is a tree obtained by applying the (inductive) rules of the type system $\mathcal{X}$. We write $\Phi \triangleright_\mathcal{X} \Gamma \vdash t : \sigma$ if Φ is a type derivation concluding with the type judgment $\Gamma \vdash t : \sigma$, and just $\triangleright_\mathcal{X} \Gamma \vdash t : \sigma$ if there exists Φ such that $\Phi \triangleright_\mathcal{X} \Gamma \vdash t : \sigma$. A term t is **$\mathcal{X}$-typable** iff there is a derivation in $\mathcal{X}$ typing t, *i.e.* if there is Φ such that $\Phi \triangleright_\mathcal{X} \Gamma \vdash t : \sigma$. We may omit the index $\mathcal{X}$ if the name of the system is clear from the context.

$$\frac{}{x : [\tau] \vdash x : \tau}(\mathtt{ax}) \qquad \frac{\Gamma \vdash t : \tau}{\Gamma \setminus\!\setminus x \vdash \lambda x.t : \Gamma(x) \Rightarrow \tau}(\Rightarrow_{\mathtt{i}}) \qquad \frac{\Gamma \vdash t : [\sigma_k]_{k\in K} \Rightarrow \tau \quad (\Gamma_k \vdash u : \sigma_k)_{k\in K}}{\Gamma \wedge_{k\in K} \Gamma_k \vdash tu : \tau}(\Rightarrow_{\mathtt{e}})$$

Figure 1 System $\mathcal{H}_\lambda$.

$$\text{Rule } (\mathtt{ax}) \qquad \text{Rule } (\Rightarrow_{\mathtt{i}}) \qquad \frac{(\Gamma_k \vdash t : \sigma_k)_{k\in K}}{\wedge_{k\in K}\Gamma_k \Vdash t : [\sigma_k]_{k\in K}}(\wedge) \qquad \frac{\Gamma \vdash t : \mathcal{I} \Rightarrow \sigma \quad \Gamma' \Vdash u : \mathcal{I}}{\Gamma \wedge \Gamma' \vdash t\, u : \sigma}(\Rightarrow_{\mathtt{e}})$$

Figure 2 System $\mathcal{H'}_\lambda$.

3.1 Characterizing Head β-Normalizing λ-Terms

We discuss in this section typing systems being able to characterize head β-normalizing λ-terms. We first consider system $\mathcal{H}_\lambda$ in Fig. 1, first appearing in [19], then in [14].

Notice that $K = \emptyset$ in rule $(\Rightarrow_{\mathtt{e}})$ allows to type an application tu without necessarily typing the subterm u. Thus for example, if $\Omega = (\lambda x.xx)(\lambda x.xx)$, then from the judgment $x : [\sigma] \vdash x : \sigma$ we can derive $x : [\sigma] \vdash (\lambda y.x)\Omega : \sigma$.

System $\mathcal{H}_\lambda$ characterizes head β-normalization:

▶ **Lemma 2.** *Let $t \in \mathcal{T}_\lambda$. Then t is $\mathcal{H}_\lambda$-typable iff $t \in \mathcal{HN}(\lambda)$.*

Moreover, the implication *typability implies normalization* can be shown by simple arithmetical arguments provided by the *quantitative* flavour of the typing system $\mathcal{H}_\lambda$, in contrast to classical reducibility arguments usually invoked in other cases [20, 29]. Actually, the arithmetical arguments give the following quantitative property:

▶ **Lemma 3.** *If t is $\mathcal{H}_\lambda$-typable with tree derivation Π, then the size (number of nodes) of Π gives an upper bound to the length of the head-reduction strategy starting at t.*

To reformulate system $\mathcal{H}_\lambda$ in a different way, we now distinguish two sorts of judgments: **regular judgments** of the form $\Gamma \vdash t : \sigma$ assign *types* to terms, and **auxiliary judgments** of the form $\Gamma \Vdash t : \mathcal{I}$ assign *intersection types* to terms.

An equivalent formulation of system $\mathcal{H}_\lambda$, called $\mathcal{H}'_\lambda$, is given in Fig. 2. There are two inherited forms of type derivations: **regular** (resp. **auxiliary**) **derivations** are those that conclude with regular (resp. auxiliary) judgments. Notice that $I = \emptyset$ in rule $(\wedge)$ gives $\Vdash u : [\,]$ for *any* term u, *e.g.* $\Vdash \Omega : [\,]$, so that one can derive $x : [\tau] \vdash (\lambda y.x)\Omega : \tau$ in this system. Notice also that systems $\mathcal{H}_\lambda$ and $\mathcal{H}'_\lambda$ are *relevant*, *i.e.* they lack weakening. Equivalence between $\mathcal{H}_\lambda$ and $\mathcal{H}'_\lambda$ gives the following result:

▶ **Corollary 4.** *Let $t \in \mathcal{T}_\lambda$. Then t is $\mathcal{H}'_\lambda$-typable iff $t \in \mathcal{HN}(\lambda)$.*

Auxiliary judgments turn out to substantially lighten the notations and to make the statements (and their proofs) more readable.

3.2 Characterizing Strong β-Normalizing λ-Terms

We now discuss typing systems being able to characterize strong β-normalizing λ-terms. We first consider system $\mathcal{S}_\lambda$ in Fig. 3, which appears in [8] (slight variants appear in [13, 6, 24]).

$$\frac{}{x : [\tau] \vdash x : \tau}\ (\mathtt{ax}) \qquad \frac{\Gamma \vdash t : \tau}{\Gamma \setminus\!\setminus x \vdash \lambda x.t : \Gamma(x) \Rightarrow \tau}\ (\Rightarrow_{\mathtt{i}}) \qquad \frac{\Gamma \vdash t : [\,] \Rightarrow \tau \qquad \Delta \vdash u : \sigma}{\Gamma \wedge \Delta \vdash tu : \tau}\ (\Rightarrow_{\mathtt{e}_1})$$

$$\frac{\Gamma \vdash t : [\sigma_k]_{k\in K} \Rightarrow \tau \quad (\Delta_k \vdash u : \sigma_k)_{k\in K} \quad K \neq \emptyset}{\Gamma \wedge_{k\in K} \Delta_k \vdash tu : \tau}\ (\Rightarrow_{\mathtt{e}_2})$$

Figure 3 System $\mathcal{S}_\lambda$.

$$\text{Rule } (\mathtt{ax}) \qquad \text{Rule } (\Rightarrow_{\mathtt{i}}) \qquad \frac{(\Gamma_k \vdash t : \sigma_k)_{k\in K}}{\wedge_{k\in K}\Gamma_k \Vdash t : [\sigma_k]_{k\in K}}\ (\wedge) \qquad \frac{\Gamma \vdash t : \mathcal{I} \Rightarrow \tau \quad \Delta \Vdash u : \mathcal{I}^*}{\Gamma \wedge \Delta \vdash tu : \tau}\ (\Rightarrow_{\mathtt{e}})$$

Figure 4 System $\mathcal{S}'_\lambda$.

Rule $(\Rightarrow_{\mathtt{e}_1})$ forces the *erasable arguments* (the subterm u) to be typed, even if the type of u (*i.e.* σ) is not being used in the conclusion of the judgment. Thus, in contrast to system $\mathcal{H}_\lambda$, every subterm of a typed term is now typed.

System $\mathcal{S}_\lambda$ characterizes strong β-normalization:

▶ **Lemma 5.** *Let $t \in \mathcal{T}_\lambda$. Then t is $\mathcal{S}_\lambda$-typable iff $t \in \mathcal{SN}(\lambda)$.*

As before, the implication *typability implies normalization* can be show by simple arithmetical arguments provided by the *quantitative* flavour of the typing system $\mathcal{S}_\lambda$.

An equivalent formulation of system $\mathcal{S}_\lambda$, called $\mathcal{S}'_\lambda$, is given in Fig. 4. As before, we use regular as well as auxiliary judgments. Notice that $I = \emptyset$ in rule $(\wedge)$ is still possible, but derivations of the form $\Vdash t : [\,]$, representing untyped terms, will never be used. The choice operation $\_^*$ (defined at the beginning of Sec. 3) in rule $(\Rightarrow_{\mathtt{e}})$ is used to impose an arbitrary type for an erasable term, *i.e.* when t has type $[\,] \Rightarrow \tau$, then u needs to be typed with an arbitrary type $[\sigma]$, thus the auxiliary judgment typing u on the right premise of $(\Rightarrow_{\mathtt{e}})$ cannot assign $[\,]$ to u. This should be understood as a sort of *controlled weakening*. Here is an example of type derivation in system $\mathcal{S}'_\lambda$:

$$\dfrac{\dfrac{\dfrac{}{x : [\sigma] \vdash x : \sigma}}{x : [\sigma] \vdash \lambda y.x : [\,] \Rightarrow \sigma} \qquad \dfrac{\dfrac{}{z : [\tau] \vdash z : \tau}}{z : [\tau] \Vdash z : [\tau]}}{x : [\sigma], z : [\tau] \vdash (\lambda y.x)z : \sigma}$$

Since $\mathcal{S}_\lambda$ and $\mathcal{S}'_\lambda$ are equivalent, we also have:

▶ **Corollary 6.** *Let $t \in \mathcal{T}_\lambda$. Then t is $\mathcal{S}'_\lambda$-typable iff $t \in \mathcal{SN}(\lambda)$.*

4 Quantitative Type Systems for the λ_μ-Calculus

We present in this section two quantitative systems for the λ_μ-calculus, systems $\mathcal{H}_{\lambda_\mu}$ (Sec. 4.2) and $\mathcal{S}_{\lambda_\mu}$ (Sec. 4.3), characterizing, respectively, head and strong λ_μ-normalizing objects. Since λ-calculus is embedded in the λ_μ-calculus, then the starting points to design $\mathcal{H}_{\lambda_\mu}$ and $\mathcal{S}_{\lambda_\mu}$ are, respectively, $\mathcal{H}'_\lambda$ and $\mathcal{S}'_\lambda$, introduced in Sec. 3.

4.1 Types

We consider a countable set of **base types** $a, b, c \dots$ and the following categories of types:

(**Object Types**)	$\mathcal{A}$	$:=$	$\mathcal{C} \mid \mathcal{U}$
(**Command Type**)	$\mathcal{C}$	$:=$	$\#$
(**Union Types**)	$\mathcal{U}, \mathcal{V}$	$::=$	$\langle \sigma_k \rangle_{k \in K}$
(**Intersection Types**)	$\mathcal{I}$	$::=$	$[\mathcal{U}_k]_{k \in K}$
(**Types**)	σ, τ	$::=$	$a \mid \mathcal{I} \Rightarrow \mathcal{U}$

The constant $\#$ is used to type commands, union types to type terms and intersection types to type variables (thus left-hand sides of arrows). Both $[\sigma_k]_{k \in \{1..n\}}$ and $\langle \sigma_k \rangle_{k \in \{1..n\}}$ can be seen as *multisets*, representing, respectively, $\sigma_1 \cap \ldots \cap \sigma_n$ and $\sigma_1 \cup \ldots \cup \sigma_n$, where $\cap$ and $\cup$ are both associative, commutative, but *non-idempotent*. We may omit the indices in the simplest case: thus $[\mathcal{U}]$ and $\langle \sigma \rangle$ denote singleton multisets. We define the operator $\wedge$ (resp. $\vee$) on intersection (resp. union) multiset types by : $[\mathcal{U}_k]_{k \in K} \wedge [\mathcal{V}_\ell]_{\ell \in L} := [\mathcal{U}_k]_{k \in K} + [\mathcal{V}_\ell]_{\ell \in L}$ and $\langle \sigma_k \rangle_{k \in K} \vee \langle \tau_\ell \rangle_{\ell \in L} := \langle \sigma_k \rangle_{k \in K} + \langle \tau_\ell \rangle_{\ell \in L}$, where $+$ always means multiset union. The *non-deterministic* **choice** operation $\_^*$ is now defined on intersection *and* union types:

$$[\mathcal{U}_k]^*_{k \in K} := \begin{cases} [\mathcal{U}] & \text{if } K = \emptyset \text{ and } \mathcal{U} \text{ is any arbitrary h-union type} \\ [\mathcal{U}_k]_{k \in K} & \text{if } K \neq \emptyset \end{cases}$$

$$\langle \sigma_k \rangle^*_{k \in K} := \begin{cases} \langle \sigma \rangle & \text{if } K = \emptyset \text{ and } \sigma \text{ is any arbitrary type} \\ [\sigma_k]_{k \in K} & \text{if } K \neq \emptyset \end{cases}$$

where an h-union type is a union type which only contains empty intersection types on the left-hand sides of arrows, *i.e.* if $\mathcal{I} \Rightarrow \mathcal{U}$ occurs in the h-union type, then $\mathcal{I} = [\,]$. The choice operator for union type is defined so that (1) the empty union cannot be assigned to μ-abstractions (see discussion on top of page 9) (2) subject reduction is guaranteed in system $\mathcal{H}_{\lambda_\mu}$ for erasing steps $(\mu\alpha.c)u \to \mu\alpha.c$ ($\alpha \notin \mathtt{fn}(c)$).

The **arity** of types and union multiset types is defined by induction: for types σ, if $\sigma = \mathcal{I} \Rightarrow \mathcal{U}$, then $\mathtt{ar}(\sigma) := \mathtt{ar}(\mathcal{U}) + 1$, otherwise, $\mathtt{ar}(\sigma) := 0$; for union multiset types, $\mathtt{ar}(\langle \sigma_k \rangle_{k \in K}) := \Sigma_{k \in K}\ \mathtt{ar}(\sigma_k)$. The **cardinality of multisets** is defined by $|[\mathcal{U}_k]_{k \in K}| = |\langle \sigma_k \rangle_{k \in K}| := |K|$.

Variable assignments (Γ), are, as before, functions from variables to intersection multiset types. Similarly, **name assignments** (Δ), are functions from names to union multiset types. The **domain of** Δ is given by $\mathtt{dom}(\Delta) := \{\alpha \mid \Delta(x) \neq \langle\,\rangle\}$, where $\langle\,\rangle$ is the empty union multiset. We may write $\emptyset$ to denote the name assignment that associates the empty union type $\langle\,\rangle$ to every name. When $\alpha \notin \mathtt{dom}(\Delta)$, then $\Delta(x)$ stands for $\langle\,\rangle$. We write $\Delta \vee \Delta'$ for $\alpha \mapsto \Delta(\alpha) + \Delta'(\alpha)$, where $\mathtt{dom}(\Delta \vee \Delta') = \mathtt{dom}(\Delta) \cup \mathtt{dom}(\Delta')$. When $\mathtt{dom}(\Gamma)$ and $\mathtt{dom}(\Gamma')$ are disjoint we may write $\Gamma; \Gamma'$ instead of $\Gamma \wedge \Gamma'$. We write $x : [\mathcal{U}_k]_{k \in K}; \Gamma$, even when $K = \emptyset$, for the following variable assignment $(x : [\mathcal{U}_k]_{k \in K}; \Gamma)(x) = [\mathcal{U}_k]_{k \in K}$ and $(x : [\mathcal{U}_k]_{k \in K}; \Gamma)(y) = \Gamma(y)$ if $y \neq x$. Similar concepts apply to name assignments, so that $\alpha : \langle \sigma_k \rangle_{k \in K}; \Delta$ and $\Delta \setminus\!\setminus \alpha$ are defined as expected.

We now present our typing systems $\mathcal{H}_{\lambda_\mu}$ and $\mathcal{S}_{\lambda_\mu}$, both having **regular** (resp. **auxiliary**) judgments of the form $\Gamma \vdash t : \mathcal{U} \mid \Delta$ (resp. $\Gamma \Vdash t : \mathcal{I} \mid \Delta$), together with their respective notions of regular and auxiliary derivations. An important syntactical property they enjoy is that both are **syntax directed**, *i.e.* for each (regular/auxiliary) typing judgment j there is a *unique* typing rule whose conclusion matches the judgment j. This makes our proofs much simpler than those arising with idempotent types which are based on long generation lemmas (*e.g.* [6, 36]).

$$\frac{\mathcal{U} \neq \langle\,\rangle}{x : [\mathcal{U}] \vdash x : \mathcal{U} \mid \emptyset}\ (\mathtt{ax}) \qquad \frac{\Gamma \vdash t : \mathcal{U} \mid \Delta}{\Gamma \setminus\!\setminus x \vdash \lambda x.t : \langle \Gamma(x) \Rightarrow \mathcal{U} \rangle \mid \Delta}\ (\Rightarrow_{\mathtt{i}}) \qquad \frac{\Gamma \vdash t : \mathcal{U} \mid \Delta}{\Gamma \vdash [\alpha]t : \# \mid \Delta \vee \{\alpha : \mathcal{U}\}}\ (\#_i)$$

$$\frac{\Gamma \vdash c : \# \mid \Delta}{\Gamma \vdash \mu\alpha.c : \Delta(\alpha)^* \mid \Delta \setminus\!\setminus \alpha}\ (\#_e) \qquad \frac{(\Gamma_k \vdash t : \mathcal{U}_k \mid \Delta_k)_{k\in K}}{\wedge_{k\in K}\Gamma_k \Vdash t : [\mathcal{U}_k]_{k\in K} \mid \vee_{k\in K}\Delta_k}\ (\wedge)$$

$$\frac{\Gamma_t \vdash t : \langle \mathcal{I}_k \Rightarrow \mathcal{U}_k \rangle_{k\in K} \mid \Delta_t \quad \Gamma_u \Vdash u : \wedge_{k\in K}\mathcal{I}_k \mid \Delta_u}{\Gamma_t \wedge \Gamma_u \vdash t\,u : \vee_{k\in K}\mathcal{U}_k \mid \Delta_t \vee \Delta_u}\ (\Rightarrow_{\mathtt{e}})$$

Figure 5 System $\mathcal{H}_{\lambda_\mu}$.

4.2 System $\mathcal{H}_{\lambda_\mu}$

In this section we present a quantitative typing system for λ_μ, called $\mathcal{H}_{\lambda_\mu}$, characterizing head λ_μ-normalization. It can be seen as a first intuitive step to understand the typing system $\mathcal{S}_{\lambda_\mu}$, introduced later in Sec. 4.3, and characterizing strong λ_μ-normalization. However, the two systems will not be described and studied in the same way: by lack of space we choose to discuss $\mathcal{H}_{\lambda_\mu}$ in a more informal and compact way, while reserving more space and discussion to system $\mathcal{S}_{\lambda_\mu}$.

The (syntax directed) rules of the typing system $\mathcal{H}_{\lambda_\mu}$ appear in Fig. 5.

Rule $(\Rightarrow_{\mathtt{e}})$ is to be understood as a *logical admissible* rule: if union (resp. intersection) is interpreted as the $\mathtt{OR}$ (resp. $\mathtt{AND}$) logical connective, then $\mathtt{OR}_{k\in K}\ (\mathcal{I}_k \Rightarrow \mathcal{U}_k)$ and $(\mathtt{AND}_{k\in K}\ \mathcal{I}_k)$ implies $(\mathtt{OR}_{k\in K}\ \mathcal{U}_k)$. As in the simply typed λ_μ-calculus [32], the $(\#_i)$ rule saves a type $\mathcal{U}$ for the name α, however, in our system, the corresponding name assignment $\Delta \vee \{\alpha : \mathcal{U}\}$, specified by means of $\vee$, collects *all* the types that α has been assigned during the derivation. Notice that the $(\#_e)$ rule is not deterministic since $\Delta(\alpha)^*$ denotes an arbitrary union type, a choice that is now discussed.

In simply typed λ_μ, **call-cc** $= \lambda y.\mu\alpha.[\alpha]y(\lambda x.\mu\beta.[\alpha]x)$ would be typed with $((a \Rightarrow b) \Rightarrow a) \Rightarrow a$ (Peirce's Law), so that the fact that α is *used* twice in the type derivation would not be explicitely materialized (same comment applies to idempotent intersection/union types). This makes a strong contrast with the derivation in Fig. 6, where $\mathcal{U}_a := \langle a \rangle$, $\mathcal{U}_b := \langle b \rangle$, $\mathcal{U}_y := \langle [\langle [\mathcal{U}_a] \Rightarrow \mathcal{U}_b \rangle] \Rightarrow \mathcal{U}_a \rangle$ and $\Phi_y \triangleright y : [\mathcal{U}_y] \vdash y : \mathcal{U}_y \mid\ .$

Indeed, we can distinguish two different uses of names :

- The name α is saved twice by a $(\#_i)$ rule : once for x and once for $y(\lambda x.\mu\beta.[\alpha]x)$, both times with type $\mathcal{U}_a$. After that, the abstraction $\mu\alpha.[\alpha]y(\lambda x.\mu\beta.[\alpha]x)$ **restores** the types that were previoulsy stored by α. A similar phenomenon occurs with λ-abstractions, which restore the types of the free ocurrences of variables in the body of the functions.
- The name β is not free in $[\alpha]x$, so that a new union type $\mathcal{U}_b$ is introduced to type the abstraction $\mu\beta.[\alpha]x$. From a logical point of view this corresponds to a *weakening* on the right handside of the sequent. Consequently, λ and μ-abstractions are not treated symmetrically: when x is not free in t, then $\lambda x.t$ will be typed with $[\,] \Rightarrow \sigma$ (where σ is the type of t), and no new intersection type is introduced for the abstracted variable x.

Thus, μ-abstractions have two uses: to restore saved types and to create new types, which explains the fact that empty union types are banned. Indeed, if $\triangleright\Gamma \vdash t : \mathcal{U} \mid \Delta$, then $\mathcal{U} \neq \langle\,\rangle$.

Why union types cannot be empty? Let us suppose that empty union types may be introduced by the $(\#_e)$ rule, at least when $\alpha \notin \mathtt{fn}(c)$, so that for example $t = \mu\beta.[\alpha]x$ would be typed with $\langle\,\rangle$ (this can be obtained by simply changing $\Delta(\alpha)^*$ to $\Delta(\alpha)$ in the $(\#_e)$-rule).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\mathcal{U}_a] \vdash x : \mathcal{U}_a \mid}}{x : [\mathcal{U}_a] \vdash [\alpha]x : \# \mid \alpha : \mathcal{U}_a}}{x : [\mathcal{U}_a] \vdash \mu\beta.[\alpha]x : \mathcal{U}_b \mid \alpha : \mathcal{U}_a}}{\vdash \lambda x.\mu\beta.[\alpha]x : \langle [\mathcal{U}_a] \Rightarrow \mathcal{U}_b \rangle \mid \alpha : \mathcal{U}_a}}{\Vdash \lambda x.\mu\beta.[\alpha]x : [\langle [\mathcal{U}_a] \Rightarrow \mathcal{U}_b \rangle] \mid \alpha : \mathcal{U}_a}}{y : [\mathcal{U}_y] \vdash y(\lambda x.\mu\beta.[\alpha]x) : \mathcal{U}_a \mid \alpha : \mathcal{U}_a}}{y : [\mathcal{U}_y] \vdash [\alpha]y(\lambda x.\mu\beta.[\alpha]x) : \# \mid \alpha : \langle \mathcal{U}_a, \mathcal{U}_a \rangle}}{y : [\mathcal{U}_y] \vdash \mu\alpha.[\alpha]y(\lambda x.\mu\beta.[\alpha]x) : \langle \mathcal{U}_a, \mathcal{U}_a \rangle \mid}}{\vdash \lambda y.\mu\alpha.[\alpha]y(\lambda x.\mu\beta.[\alpha]x) : \langle [\mathcal{U}_y] \Rightarrow \langle \mathcal{U}_a, \mathcal{U}_a \rangle \rangle \mid}$$

Φ_y

Figure 6 Typing **call-cc**.

Suppose also an object o containing 2 occurrences of the subterm $[\gamma]t$, so that γ receives the union type $\langle\,\rangle$ twice in the corresponding name assignment. Then, the term $\mu\gamma.o$ will be typed with $\langle\,\rangle = \langle\,\rangle \vee \langle\,\rangle$, which does not reflect the fact that γ is used twice, thus loosing the *quantitative* flavour of the system (see also a formal argument just after Lem. 9).

We define now the notion of **size derivation**, which is a natural number representing the amount of information in a tree derivation. For any type derivation Φ, $\mathtt{sz}(\Phi)$ is inductively defined by the following rules, where we use an abbreviated notation for the premises.

$$\mathtt{sz}\left(\dfrac{}{x : [\mathcal{U}] \vdash x : \mathcal{U} \mid \emptyset}\ (\mathtt{ax})\right) := 1$$

$$\mathtt{sz}\left(\dfrac{\Phi_t \rhd t}{\Gamma \setminus\!\setminus x \vdash \lambda x.t : \langle \Gamma(x) \Rightarrow \mathcal{U} \rangle \mid \Delta}\ (\Rightarrow_{\mathtt{i}})\right) := \mathtt{sz}(\Phi_t) + 1$$

$$\mathtt{sz}\left(\dfrac{\Phi_t \rhd t}{\Gamma \vdash [\alpha]t : \# \mid \Delta \vee \{\alpha : \mathcal{U}\}}\ (\#_i)\right) := \mathtt{sz}(\Phi_t) + \mathtt{ar}(\mathcal{U})$$

$$\mathtt{sz}\left(\dfrac{\Phi_c \rhd c}{\Gamma \vdash \mu\alpha.c : \Delta(\alpha)^* \mid \Delta \setminus\!\setminus \alpha}\ (\#_e)\right) := \mathtt{sz}(\Phi_c) + 1$$

$$\mathtt{sz}\left(\dfrac{(\Phi_k \rhd t)_{k \in K}}{\wedge_{k \in K}\Gamma_k \Vdash t : [\mathcal{U}_k]_{k \in K} \mid \vee_{k \in K}\Delta_k}\ (\wedge)\right) := \Sigma_{k \in K}\ \mathtt{sz}(\Phi_k)$$

$$\mathtt{sz}\left(\dfrac{\Phi_t \rhd t \qquad \Phi_u \rhd u}{\Gamma \vdash t\,u : \vee_{k \in K}\mathcal{V}_k \mid \Delta}\ (\Rightarrow_{\mathtt{e}})\right) := \mathtt{sz}(\Phi_t) + \mathtt{sz}(\Phi_u) + |K|$$

System $\mathcal{H}_{\lambda_\mu}$ behaves as expected, in particular, typing is stable by reduction (Subject Reduction) and anti-reduction (Subject Expansion). Moreover,

▶ **Theorem 7.** *Let $o \in \mathcal{O}_{\lambda_\mu}$. Then o is $\mathcal{H}_{\lambda_\mu}$-typable iff $o \in \mathcal{HN}(\lambda_\mu)$ iff the head-strategy terminates on o. Moreover, if o is $\mathcal{H}_{\lambda_\mu}$-typable with tree derivation Π, then $\mathtt{sz}(\Pi)$ gives an upper bound to the length of the head-reduction strategy starting at o.*

We do not provide the proof of this theorem, because it uses special cases of the more general technology that we are going to develop later to deal with strong normalization. Notice that Thm. 7 ensures that the head-strategy is complete for head-normalization in λ_μ.

A last comment of this section concerns the restriction of system $\mathcal{H}_{\lambda_\mu}$ to the pure λ-calculus: union types, name assignments and rules $(\#_e)$ and $(\#_i)$ are no more necessary, so that every union multiset takes the single form $\langle\tau\rangle$, which can be simply identified with τ. Thus, the restricted typing system $\mathcal{H}_{\lambda_\mu}$ becomes the one in Fig. 2.

$$\frac{\mathcal{U} \neq \langle\,\rangle}{x : [\mathcal{U}] \vdash x : \mathcal{U} \mid \emptyset}\ (\mathtt{ax}) \qquad \frac{\Gamma \vdash t : \mathcal{U} \mid \Delta}{\Gamma \setminus\!\setminus x \vdash \lambda x.t : \langle \Gamma(x) \Rightarrow \mathcal{U} \rangle \mid \Delta}\ (\Rightarrow_{\mathtt{i}}) \qquad \frac{\Gamma \vdash t : \mathcal{U} \mid \Delta}{\Gamma \vdash [\alpha]t : \# \mid \Delta \vee \{\alpha : \mathcal{U}\}}\ (\#_i)$$

$$\frac{\Gamma \vdash c : \# \mid \Delta}{\Gamma \vdash \mu\alpha.c : \Delta(\alpha)^* \mid \Delta \setminus\!\setminus \alpha}\ (\#_e) \qquad \frac{(\Gamma_k \vdash t : \mathcal{U}_k \mid \Delta_k)_{k \in K}}{\wedge_{k \in K} \Gamma_k \Vdash t : [\mathcal{U}_k]_{k \in K} \mid \vee_{k \in K} \Delta_k}\ (\wedge)$$

$$\frac{\Gamma_t \vdash t : \langle \mathcal{I}_k \Rightarrow \mathcal{U}_k \rangle_{k \in K} \mid \Delta_t \qquad \Gamma_u \Vdash u : \wedge_{k \in K}\ (\mathcal{I}_k^*) \mid \Delta_u}{\Gamma_t \wedge \Gamma_u \vdash t\,u : \vee_{k \in K} \mathcal{U}_k \mid \Delta_t \vee \Delta_u}\ (\Rightarrow_{\mathtt{e}})$$

Figure 7 System $\mathcal{S}_{\lambda_\mu}$.

4.3 System $\mathcal{S}_{\lambda_\mu}$

This section presents a quantitative typing system characterizing strongly β-normalizing λ_μ-terms. The (syntax directed) typing rules of the typing system $\mathcal{S}_{\lambda_\mu}$ appear in Fig. 7.

As in system $\mathcal{S}'_\lambda$, the operation $\_^*$ is used to choose arbitrary types for erasable terms, so that no subterm is untyped, thus ensuring strong λ_μ-normalization. While the use of $\_^*$ in the $(\#_e)$-rule can be seen as a *weakening* on the right hand-sides of sequents, its use in rule $(\Rightarrow_{\mathtt{e}})$ corresponds to a form of *controlled weakening* on the left hand-sides. We still consider the definition of size given before, as the choice operator does not play any particular role.

As in system $\mathcal{H}_{\lambda_\mu}$, a term is typed with a non-empty union type:

▶ **Lemma 8.** *If* $\triangleright\Gamma \vdash t : \mathcal{U} \mid \Delta$, *then* $\mathcal{U} \neq \langle\,\rangle$.

As well as in the case of $\mathcal{H}_{\lambda_\mu}$, system $\mathcal{S}_{\lambda_\mu}$ can be restricted to the pure λ-calculus. Using the same observations at the end of Sec. 4.2 we obtain the typing system $\mathcal{S}'_\lambda$ in Fig. 4 that characterizes β-strong normalization.

A key property of system $\mathcal{S}_{\lambda_\mu}$ is known as *relevance*:

▶ **Lemma 9** (Relevance). *If* $\Phi \triangleright \Gamma \vdash o : \mathcal{A} \mid \Delta$, *then* $\mathtt{dom}(\Gamma) = \mathtt{fv}(o)$ *and* $\mathtt{dom}(\Delta) = \mathtt{fn}(o)$.

Relevance holds thanks to the choice operator $\_^*$: indeed, if $\Delta(\alpha)^*$ is replaced by $\Delta(\alpha)$ in the $(\#_e)$-rule, then the following derivations gives a counter-example to the relevance property, where $\alpha \in \mathtt{fn}([\alpha]\mu\beta.[\gamma]x)$ but $\alpha \notin \mathtt{dom}(\gamma : \langle a \rangle)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\langle a \rangle] \vdash x : \langle a \rangle \mid}}{x : [\langle a \rangle] \vdash [\gamma]x : \# \mid \gamma : \langle a \rangle}}{x : [\langle a \rangle] \vdash \mu\beta.[\gamma]x : \langle\,\rangle \mid \gamma : \langle a \rangle}}{x : [\langle a \rangle] \vdash [\alpha]\mu\beta.[\gamma]x : \# \mid \gamma : \langle a \rangle}$$

Indeed, the size of derivations typing commands takes into account the arity of their corresponding type; and this is essential to materialize a decreasing measure for μ-reduction (see Sec. 5). Notice that $\mathtt{sz}(\Phi) \geq 1$ holds for any *regular* derivation Φ, whereas, by definition, the derivation of the empty auxiliary judgment $\Vdash t : [\,] \mid$ has size 0.

5 Typing Properties

This section shows two fundamental properties of reduction (*i.e. forward*) and anti-reduction (*i.e. backward*) of system $\mathcal{S}_{\lambda_\mu}$. In Sec. 5.1 we analyse the *subject reduction (SR)* property, and we prove that reduction preserves typing and decreases the size of type derivations (that is why

we call it weighted SR). The proof of this property makes use of two fundamental properties (Lem. 11 and 12) guaranteeing well-typedness of the meta-operations of substitution and replacement. Sec. 5.2 is devoted to *subject expansion (SE)*, which states that non-erasing anti-reduction preserves types. The proof uses the fact that reverse substitution (Lem. 13) and reverse replacement (Lem. 14) preserve types.

We start by stating an interesting property, to be used in our forthcoming lemmas, that allows us to split and merge auxilary derivations:

▶ **Lemma 10.** *Let* $\mathcal{I} = \wedge_{k\in K}\mathcal{I}_k$. *Then* $\Phi \triangleright \Gamma \Vdash t : \mathcal{I} \mid \Delta$ *iff* $\exists(\Gamma_k)_{k\in K}, \exists(\Delta_k)_{k\in K}$ *s.t.* $(\Phi_k \triangleright \Gamma_k \Vdash t : \mathcal{I}_k \mid \Delta_k)_{k\in K}$, $\Gamma = \wedge_{k\in K}\Gamma_k$ *and* $\Delta = \vee_{k\in K}\Delta_k$. *Moreover,* $\mathtt{sz}(\Phi) = \Sigma_{k\in K}\mathtt{sz}(\Phi_k)$.

5.1 Forward Properties

We first state the substitution lemma, which guarantees that typing is stable by substitution. The lemma also establishes the size of the derivation tree of a substituted object from the sizes of the derivations trees of its components.

▶ **Lemma 11** (Substitution). *Let* $\Theta_u \triangleright \Gamma_u \Vdash u : \mathcal{I} \mid \Delta_u$. *If* $\Phi_o \triangleright \Gamma; x : \mathcal{I} \vdash o : \mathcal{A} \mid \Delta$, *then there is* $\Phi_{o\{x/u\}}$ *such that*

- $\Phi_{o\{x/u\}} \triangleright \Gamma \wedge \Gamma_u \vdash o\{x/u\} : \mathcal{A} \mid \Delta \vee \Delta_u$.
- $\mathtt{sz}\left(\Phi_{o\{x/u\}}\right) = \mathtt{sz}(\Phi_o) + \mathtt{sz}(\Theta_u) - |\mathcal{I}|$.

Proof. By induction on Φ_o using Lem. 9 and 10. ◀

Typing is also stable by replacement. Moreover, we can specify the exact size of the derivation tree of the replaced object from the sizes of its components.

▶ **Lemma 12** (Replacement). *Let* $\Theta_u \triangleright \Gamma_u \Vdash u : \wedge_{k\in K}\ (\mathcal{I}_k^*) \mid \Delta_u$ *where* $\alpha \notin \mathtt{fn}(u)$. *If* $\Phi_o \triangleright \Gamma \vdash o : \mathcal{A} \mid \alpha : \langle \mathcal{I}_k \Rightarrow \mathcal{V}_k\rangle_{k\in K}; \Delta$, *then there is* $\Phi_{o\{\alpha/\!\!/u\}}$ *such that :*

- $\Phi_{o\{\alpha/\!\!/u\}} \triangleright \Gamma \wedge \Gamma_u \vdash o\{\alpha/\!\!/u\} : \mathcal{A} \mid \alpha : \vee_{k\in K}\mathcal{V}_k; \Delta \vee \Delta_u$.
- $\mathtt{sz}\left(\Phi_{o\{\alpha/\!\!/u\}}\right) = \mathtt{sz}(\Phi_o) + \mathtt{sz}(\Theta_u)$.

Proof. By induction on Φ using Lem. 9 and 10. ◀

Notice that the type of α in the conclusion of the derivation $\Phi_{o\{\alpha/\!\!/u\}}$ (which is $\vee_{k\in K}\mathcal{V}_k$) is strictly smaller than that of the conclusion of the derivation Φ_o (which is $\langle \mathcal{I}_k \Rightarrow \mathcal{V}_k\rangle_{k\in K}$) if and only if $K \neq \emptyset$.

Lemmas 11 and 12 are used in the proof of the following key property.

▶ **Property 1** (Weighted Subject Reduction for λ_μ). *Let* $\Phi \triangleright \Gamma \vdash o : \mathcal{A} \mid \Delta$. *If* $o \to o'$ *is a non-erasing step, then there exists a derivation* $\Phi' \triangleright \Gamma \vdash o' : \mathcal{A} \mid \Delta$ *such that* $\mathtt{sz}(\Phi) > \mathtt{sz}(\Phi')$.

Proof. By induction on $o \to o'$ using Lem. 9, 11 and 12. ◀

Discussion. A first remark about the property above is that variable and name assignments are not necessarily preserved by erasing reductions. Thus for example, consider $t = (\lambda y.x)z \to x = t'$. The term t is typed with a variable assignment whose domain is $\{x, z\}$, while t' can only be typed with an assignment whose domain is $\{x\}$. Concretely, starting from a derivation of $x : [\langle a\rangle], z : [\langle b\rangle] \vdash (\lambda x.y)z : \langle a\rangle$ (the simplified type derivation of this term in the $\mathcal{S}'_\lambda$ system appears on page 24:6), we can only construct a derivation of $x : [\langle a\rangle] \vdash x : \langle a\rangle$, so that the type is preserved while the variable assignment is not. Actually, our restricted form of subject reduction (*i.e.* for non-erasing steps only) is sufficient for our purpose (see how we deal with the erasing steps in the proof of Lem. 16).

A second remark is that the consideration of arities of names in the definition of the size of derivations (third case $(\#_i)$) is crucial to guarantee that μ-reduction decreases $\mathtt{sz}(\_)$. This is perfectly reflected in Lem. 12, where the type of α in the conclusion of the derivation $\Phi_{o\{\alpha /\!\!/ u\}}$ is strictly smaller than that of the conclusion of the derivation Φ_o.

A third point is about the use of the choice operator in the typing rule $(\#_e)$, which does not allow for the type $\langle\,\rangle$ to be assigned to α when $\alpha \notin \mathtt{fn}(c)$. More precisely, assume, just temporarily, that the $(\#_e)$ rule does not use the choice operator, so that a μ-abstraction can be typed with $\langle\,\rangle$. Set $u := \mu\beta.[\gamma]y$ and $c := [\alpha]\mu\delta.[\alpha]u$ so that u, $\mu\delta.[\alpha]u$ and $\mu\alpha.c$ are typed with $\langle\,\rangle$. The resulting type derivation $\Phi_c \triangleright \Gamma \vdash c : \# \mid \Delta$ contradicts the Relevance Lem. 9, simply because $\alpha \notin \mathtt{fn}(\Delta)$ but α has two free occurrences in c. This has heavy consequences that can be illustrated by the reduction sequence $t = (\mu\alpha.c)x \to \mu\alpha.[\alpha](\mu\delta.[\alpha](\mu\beta.[\gamma]y)x)x \to^* \mu\alpha.c = t'$. Indeed, the type of $\mu\alpha.c$, which is $\langle\,\rangle$, holds no information capturing the number of free occurrences of α in c, so that there is no local way to know how many times the argument x should be typed in the whole derivation of the term $(\mu\alpha.c)x$. This prevents the reduction relation to decrease any reasonable measure associated to type derivations.

5.2 Backward Properties

Subject expansion is based on two technical properties: the first one, called reverse substitution, allows us to extract type information for an object o and a term u from the type derivation of $o\{x/u\}$; similarly, the second one, called reverse replacement, gives type information for a command c and a term u from the type derivation of $c\{\alpha /\!\!/ u\}$. Both of them are proved by induction on derivations using Lem. 9 and 10. Formally,

▶ **Lemma 13** (Reverse Substitution). *Let* $\Phi' \triangleright \Gamma' \vdash o\{x/u\} : \mathcal{A} \mid \Delta'$ *Then* $\exists\Gamma, \exists\Delta, \exists\mathcal{I}, \exists\Gamma_u, \exists\Delta_u$ *such that:* $\Gamma' = \Gamma \wedge \Gamma_u$, $\Delta' = \Delta \vee \Delta_u$, $\triangleright\Gamma; x : \mathcal{I} \vdash o : \mathcal{A} \mid \Delta$ *and* $\triangleright\Gamma_u \Vdash u : \mathcal{I} \mid \Delta_u$.

▶ **Lemma 14** (Reverse Replacement). *Let* $\Phi' \triangleright \Gamma' \vdash o\{\alpha /\!\!/ u\} : \mathcal{A} \mid \alpha : \mathcal{V}; \Delta'$, *where* $\alpha \notin \mathtt{fn}(u)$. *Then* $\exists\Gamma, \exists\Delta, \exists\Gamma_u, \exists\Delta_u, \exists(\mathcal{I}_k)_{k\in K}, \exists(\mathcal{V}_k)_{k\in K}$ *such that:* $\Gamma' = \Gamma \wedge \Gamma_u$, $\Delta' = \Delta \vee \Delta_u$, $\mathcal{V} = \vee_{k\in K}\mathcal{V}_k$, $\triangleright\Gamma \vdash o : \mathcal{A} \mid \alpha : \langle \mathcal{I}_k \to \mathcal{V}_k\rangle_{k\in K}; \Delta$, *and and* $\triangleright\Gamma_u \Vdash u : \wedge_{k\in K}\mathcal{I}_k^* \mid \Delta_u$.

The following property will be used in Sec. 6 to show that normalization implies typability.

▶ **Property 2** (Subject Expansion for λ_μ). *Assume* $\Phi' \triangleright \Gamma' \vdash o' : \mathcal{A} \mid \Delta'$. *If* $o \to o'$ *is a non-erasing step, then there is* $\Phi \triangleright \Gamma' \vdash o : \mathcal{A} \mid \Delta'$.

Proof. By induction on $\to$ using Lem. 9, 13 and 14. ◀

6 Strongly Normalizing λ_μ-Objects

In this section we show the characterization of strongly-normalizing terms of the $\lambda\mu$-calculus by means of the typing system introduced in Sec. 4, *i.e.* we show that an object o is strongly-normalizing iff t is typable.

The proof of our main result (Thm. 18) relies on the following two ingredients:

- Every $\mathcal{S}_{\lambda_\mu}$-typable object is in $\mathcal{SN}(\lambda_\mu)$ (Lem. 16).
- Every object in $\mathcal{SN}(\lambda_\mu)$ is $\mathcal{S}_{\lambda_\mu}$-typable (Lem. 17).

First, we inductively reformulate the set of strongly normalizing objects: the set $\mathcal{I}(\lambda_\mu)$ is defined as the smallest subset of $\mathcal{O}_{\lambda_\mu}$ satisfying the following closure properties:

1. If $t_1, \ldots, t_n$ $(n \geq 0) \in \mathcal{I}(\lambda_\mu)$, then $xt_1 \ldots t_n \in \mathcal{I}(\lambda_\mu)$.
2. If $t \in \mathcal{I}(\lambda_\mu)$, then $\lambda x.t \in \mathcal{I}(\lambda_\mu)$.

3. If $c \in \mathcal{I}(\lambda_\mu)$, then $\mu\alpha.c \in \mathcal{I}(\lambda_\mu)$.
4. If $t \in \mathcal{I}(\lambda_\mu)$, then $[\alpha]t \in \mathcal{I}(\lambda_\mu)$.
5. If $u, t\{x/u\}\overline{v}$ $(n \geq 0) \in \mathcal{I}(\lambda_\mu)$, then $(\lambda x.t)u\overline{v} \in \mathcal{I}(\lambda_\mu)$.
6. If $u, (\mu\alpha.c\{\alpha /\!\!/ u\})\overline{v}$ $(n \geq 0) \in \mathcal{I}(\lambda_\mu)$, then $(\mu\alpha.c)u\overline{v} \in \mathcal{I}(\lambda_\mu)$.

The sets $\mathcal{SN}(\lambda_\mu)$ and $\mathcal{I}(\lambda_\mu)$ turn out to be equal, as expected:

▶ **Lemma 15.** $\mathcal{SN}(\lambda_\mu) = \mathcal{I}(\lambda_\mu)$.

Proof. $\mathcal{SN}(\lambda_\mu) \subseteq \mathcal{I}(\lambda_\mu)$ is proved by induction on the pair $\langle \eta(o), |o| \rangle$ endowed with the lexicographic order, where $\eta(o)$ denotes the maximal length of an λ_μ-reduction sequence starting at o and $|o|$ denotes the size of o. $\mathcal{I}(\lambda_\mu) \subseteq \mathcal{SN}(\lambda_\mu)$ is proved by induction on $\mathcal{I}(\lambda_\mu)$ using Cor. 1. No reducibility argument is then needed in this proof. ◀

▶ **Lemma 16.** *If o is $\mathcal{S}_{\lambda_\mu}$-typable, then $o \in \mathcal{SN}(\lambda_\mu)$.*

Proof. We proceed by induction on $\mathtt{sz}(\Phi)$, where $\Phi \triangleright \Gamma \vdash o : \mathcal{A} \mid \Delta$. When Φ does not end with the rule $(\Rightarrow_{\mathtt{e}})$ the proof is straightforward, so we consider Φ ends with $(\Rightarrow_{\mathtt{e}})$, where $\mathcal{A} = \mathcal{U}$ and $o = xt_1 \dots t_n$ or $o = (\mu\alpha.c)t_1 \dots t_n$ or $o = (\lambda x.u)t_1 \dots t_n$, where $n \geq 1$.
By construction there are subderivations $(\Phi_{t_i})_{i\in\{1\dots n\}}$ such that $(\mathtt{sz}(\Phi_{t_i}) < \mathtt{sz}(\Phi))_{i\in\{1\dots n\}}$ so that the *i.h.* gives $(t_i \in \mathcal{I}(\lambda_\mu))_{i\in\{1\dots n\}}$. There are three different cases:
If $o = xt_1 \dots t_n$, then from $t_i \in \mathcal{I}(\lambda_\mu)$ $(1 \leq i \leq n)$ we conclude directly $xt_1 \dots t_n \in \mathcal{I}(\lambda_\mu)$.
If $o = (\mu\alpha.c)t_1 \dots t_n$, there are two cases:

- $\alpha \in \mathtt{fn}(c)$. Using Prop. 1 we get $\Phi' \triangleright \Gamma \vdash (\mu\alpha.c\{\alpha /\!\!/ t_1\})t_2 \dots t_n : \mathcal{U} \mid \Delta$ and $\mathtt{sz}(\Phi') < \mathtt{sz}(\Phi)$. Then the *i.h.* gives $(\mu\alpha.c\{\alpha /\!\!/ t_1\})t_2 \dots t_n \in \mathcal{I}(\lambda_\mu)$. This, together with $t_1 \in \mathcal{I}(\lambda_\mu)$ gives $o \in \mathcal{I}(\lambda_\mu)$.
- $\alpha \notin \mathtt{fn}(c)$. Then it is easy to build a type derivation $\Phi' \triangleright \Gamma' \vdash (\mu\alpha.c)t_2 \dots t_n : \mathcal{U} \mid \Delta'$ verifying $\mathtt{sz}(\Phi') < \mathtt{sz}(\Phi)$, so that $(\mu\alpha.c)t_2 \dots t_n \in \mathcal{I}(\lambda_\mu)$ holds by the *i.h.* This, together with with $t_1 \in \mathcal{I}(\lambda_\mu)$ gives $o \in \mathcal{I}(\lambda_\mu)$.

If $o = (\lambda x.u)t_1 \dots t_n$, we reason similarly to the previous one. ◀

Normalization also implies typability:

▶ **Lemma 17.** *If $o \in \mathcal{SN}(\lambda_\mu)$, then o is $\mathcal{S}_{\lambda_\mu}$-typable.*

Proof. Thanks to Lem. 15 we can reason by induction on $o \in \mathcal{I}(\lambda_\mu) = \mathcal{SN}(\lambda_\mu)$. The four first cases are straightforward.

Let $o = (\lambda x.u)vt_1 \dots t_n \in \mathcal{I}(\lambda_\mu)$ coming from $u\{x/v\}t_1 \dots t_n, v \in \mathcal{I}(\lambda_\mu)$. By the *i.h.* $u\{x/v\}t_1 \dots t_n$ and v are both typable. We consider two cases. If $x \in \mathtt{fv}(u)$, then $(\lambda x.u)vt_1 \dots t_n$ is typable by Prop. 2. Otherwise, by construction, we get typing derivations for $u, t_1 \dots, t_n$ which can easily be used to build a typing derivation of $(\lambda x.u)vt_1 \dots t_n$.

Let $o = (\mu\alpha.c)vt_1 \dots t_n \in \mathcal{I}(\lambda_\mu)$ coming from $(\mu\alpha.c\{\alpha /\!\!/ v\})t_1 \dots t_n, v \in \mathcal{I}(\lambda_\mu)$. By the *i.h.* $(\mu\alpha.c\{\alpha /\!\!/ v\})t_1 \dots t_n$ and v are both typable. We consider two cases. If $\alpha \in \mathtt{fn}(c)$, then $(\mu\alpha.c)vt_1 \dots t_n$ is typable by Prop. 2. Otherwise, by construction, we get typing derivations for $c, t_1 \dots, t_n$ which can easily be used to build a typing derivation of $(\mu\alpha.c)vt_1 \dots t_n$. ◀

Lem. 16 and 17 allow us to conclude with the main result of this paper which is the equivalence between typability and strong-normalization for the λ_μ-calculus. Notice that no reducibility argument was used in the whole proof.

▶ **Theorem 18.** *Let $o \in \mathcal{O}_{\lambda_\mu}$. Then o is typable in system $\mathcal{S}_{\lambda_\mu}$ iff $o \in \mathcal{SN}(\lambda_\mu)$. Moreover, if o is $\mathcal{S}_{\lambda_\mu}$-typable with tree derivation Π, then $\mathtt{sz}(\Pi)$ gives an upper bound to the maximal length of a reduction sequence starting at o.*

To prove the second statement it is sufficient to endow the system with non-relevant axioms for variables and names. This modification, which does not recover subject expansion, is however sufficient to guarantee *weighted* subject reduction in all the cases (erasing and non-erasing steps) without changing the original measure of the derivations in system $\mathcal{S}_{\lambda_\mu}$.

7 Conclusion

This paper provides two quantitative type assignment systems $\mathcal{H}_{\lambda_\mu}$ and $\mathcal{S}_{\lambda_\mu}$ for λ_μ, characterizing, respectively, head and strongly normalizing terms. We have shown that whenever o is typable in system $\mathcal{H}_{\lambda_\mu}$, then we can extract a measure from its type derivation which provides an upper bounds to the length of the head-reduction strategy starting at o. The same happens with system $\mathcal{S}_{\lambda_\mu}$ with respect to the maximal length of a reduction sequence starting at o: indeed, the system $\mathcal{S}_{\lambda_\mu}$ endowed with weakening axioms enjoys full subject reduction (on erasing and non-erasing steps), and $\mathcal{S}_{\lambda_\mu}$ can be embedded in such a system by preserving the size of derivations.

The construction of these typing systems suggests the definition of a resource aware calculus, coming along with the corresponding extensions of the typing systems presented here, and implementing a small step operational semantics for classical natural deduction. Unfortunately we cannot provide here the details of such development due to lack of space, but they can be found in [26]. Such a calculus can be seen as an extension of the *substitution at a distance paradigm* [2, 1] to the classical case.

Quantitative types are a powerful tool to provide relational models for λ-calculus [14, 3]. The construction of such models for λ_μ should be investigated, particularly to understand in the classical case the collapse relation between quantitative and qualitative models [18].

We expect to be able to transfer the ideas in this paper to a classical sequent calculus system, as was already done for focused intuitionistic logic [25].

The fact that idempotent types were already used to show observationally equivalence between call-by-name and call-by-need [23] in intuitionistic logic suggests that our typing system $\mathcal{S}_{\lambda_{\mu r}}$ could be used in the future to understand from a semantical point of view the fact that classical call-by-name and classical call-by-need are not observationally equivalent [33].

Moreover, it is possible to obtain *exact* bounds (as in [5]) for the lengths of the head-reduction and the perpetual reduction sequences. For that, it is necessary to integrate some additional typing rules being able to type the constructors appearing in the normal forms of the terms. Although this concrete development remains as future work, the difficult and conceptual part of the technique stays in finding the decreasing measure for reduction, which is one of the contributions of this paper.

The inhabitation problem for λ-calculus is known to be undecidable for idempotent intersection types [35], but decidable for the non-idempotent ones [7]. We may conjecture that inhabitation is also decidable for $\mathcal{H}_{\lambda_\mu}$.

Acknowledgment. We would like to thank Vincent Guisse, who started a reflexion on quantitative types for the λ_μ-calculus during his M1 stage in Univ. Paris-Diderot.

References

1 Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In Peter Sewell, editor, *Proceedings of the 41st Annual ACM Symposium on Principles of Programming Languages (POPL)*, pages 659–670. ACM Press, 2014.

2 Beniamino Accattoli and Delia Kesner. The structural *lambda*-calculus. In Anuj Dawar and Helmut Veith, editors, *Proceedings of 24th EACSL Conference on Computer Science Logic*, volume 6247 of *Lecture Notes in Computer Science*, pages 381–395. Springer-Verlag, August 2010.

3 Shahin Amini and Thomas Erhard. On Classical PCF, Linear Logic and the MIX Rule. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 582–596. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

4 Zena M. Ariola, Hugo Herbelin, and Alexis Saurin. Classical call-by-need and duality. In Luke Ong [31], pages 27–44.

5 Alexis Bernadet and Stéphane Lengrand. Complexity of strongly normalising λ-terms via non-idempotent intersection types. In Martin Hofmann, editor, *Foundations of Software Science and Computation Structures (FOSSACS)*, volume 6604 of *Lecture Notes in Computer Science*, pages 88–107. Springer-Verlag, 2011.

6 Alexis Bernadet and Stéphane Lengrand. Non-idempotent intersection types and strong normalisation. *Logical Methods in Computer Science*, 9(4), 2013.

7 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. The inhabitation problem for non-idempotent intersection types. In Díaz et al. [16], pages 341–354.

8 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 2017.

9 Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type assignment for lambda-terms. *Archive for Mathematical Logic*, 19:139–156, 1978.

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

11 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.

12 René David and Karim Nour. A short proof of the strong normalization of classical natural deduction with disjunction. *J. Symb. Log.*, 68(4):1277–1288, 2003.

13 Erika De Benedetti and Simona Ronchi Della Rocca. Bounding normalization time through intersection types. In Graham-Lengrand and Paolini [21], pages 48–57.

14 Daniel de Carvalho. *Sémantiques de la logique linéaire et temps de calcul*. These de doctorat, Université Aix-Marseille II, 2007.

15 Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *CoRR*, abs/0905.4251, 2009.

16 Josep Díaz, Ivan Lanese, and Davide Sangiorgi, editors. *Proceedings of the 8th International Conference on Theoretical Computer Science (TCS)*, volume 8705 of *Lecture Notes in Computer Science*. Springer-Verlag, 2014.

17 Daniel J. Dougherty, Silvia Ghilezan, and Pierre Lescanne. Characterizing strong normalization in the curien-herbelin symmetric lambda calculus: Extending the coppo-dezani heritage. *Theoretical Computer Science*, 398(1-3):114–128, 2008.

18 Thomas Ehrhard. Collapsing non-idempotent intersection types. In Patrick Cégielski and Arnaud Durand, editors, *Proceedings of 26th EACSL Conference on Computer Science Logic*, volume 16 of *LIPIcs*, pages 259–273. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.

19 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.

20 Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge University Press, 1990.

21 Stéphane Graham-Lengrand and Luca Paolini, editors. *Proceedings of the Sixth Workshop on Intersection Types and Related Systems (ITRS), Dubrovnik, Croatia, 2012*, volume 121 of *Electronic Proceedings in Theoretical Computer Science*, 2013.

22 Timothy Griffin. A formulae-as-types notion of control. In *17th Annual ACM Symposium on Principles of Programming Languages (POPL)*, pages 47–58. ACM Press, 1990.

23 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-Verlag, 2016.

24 Delia Kesner and Daniel Ventura. Quantitative types for the linear substitution calculus. In Díaz et al. [16], pages 296–310.

25 Delia Kesner and Daniel Ventura. A resource aware computational interpretation for herbelin's syntax. In Martin Leucker, Camilo Rueda, and Frank D. Valencia, editors, *Theoretical Aspects of Computing – ICTAC 2015 – 12th International Colloquium Cali, Colombia, October 29-31, 2015, Proceedings*, volume 9399 of *Lecture Notes in Computer Science*, pages 388–403. Springer-Verlag, 2015.

26 Delia Kesner and Pierre Vial. Types as resources for classical natural deduction, 2017. Available at `http://www.irif.fr/~kesner/papers/lambda-mu.pdf`.

27 Assaf Kfoury. A linearization of the lambda-calculus and consequences. Technical report, Boston Universsity, 1996.

28 Kentaro Kikuchi and Takafumi Sakurai. A translation of intersection and union types for the $\lambda\mu$-calculus. In Jacques Garrigue, editor, *Programming Languages and Systems – 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings*, volume 8858 of *Lecture Notes in Computer Science*, pages 120–139. Springer-Verlag, 2014.

29 Jean-Louis Krivine. *Lambda-calculus, types and models*. Ellis Horwood, 1993.

30 Olivier Laurent. On the denotational semantics of the untyped lambda-mu calculus, 2004. Unpublished note.

31 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*. Springer-Verlag, 2011.

32 Michel Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In Andrei Voronkov, editor, *International Conference on Logic Programming and Automated Reasoning*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, July 1992.

33 Pierre-Marie Pédrot and Alexis Saurin. Classical by-need. In Peter Thiemann, editor, *Programming Languages and Systems – 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 616–643. Springer-Verlag, 2016.

34 Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Mathematical Structures in Computer Science*, 11(2):207–260, 2001.

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

36 Steffen van Bakel. Sound and complete typing for lambda-mu. In Elaine Pimentel, Betti Venneri, and Joe B. Wells, editors, *Proceedings Fifth Workshop on Intersection Types and*

Related Systems, ITRS 2010, Edinburgh, U.K., 9th July 2010, volume 45 of *EPTCS*, pages 31–44, 2010.

37 Steffen van Bakel, Franco Barbanera, and Ugo de'Liguoro. A filter model for the $\lambda\mu$-calculus – (extended abstract). In Luke Ong [31], pages 213–228.

38 Steffen van Bakel, Franco Barbanera, and Ugo de'Liguoro. Characterisation of strongly normalising lambda-mu-terms. In Graham-Lengrand and Paolini [21], pages 1–17.

39 Lionel Vaux. Convolution lambda-bar-mu-calculus. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4583 of *Lecture Notes in Computer Science*, pages 381–395. Springer-Verlag, 2007.

A Fibrational Framework for Substructural and Modal Logics*

Daniel R. Licata[1], Michael Shulman[2], and Mitchell Riley[3]

1 **Wesleyan University, Middletown, CT, USA**
2 **University of San Diego, San Diego, CA, USA**
3 **Wesleyan University, Middletown, CT, USA**

Abstract

We define a general framework that abstracts the common features of many intuitionistic substructural and modal logics / type theories. The framework is a sequent calculus / normal-form type theory parametrized by a *mode theory*, which is used to describe the structure of contexts and the structural properties they obey. In this sequent calculus, the context itself obeys standard structural properties, while a term, drawn from the mode theory, constrains how the context can be used. Product types, implications, and modalities are defined as instances of two general connectives, one positive and one negative, that manipulate these terms. Specific mode theories can express a range of substructural and modal connectives, including non-associative, ordered, linear, affine, relevant, and cartesian products and implications; monoidal and non-monoidal functors, (co)monads and adjunctions; n-linear variables; and bunched implications. We prove cut (and identity) admissibility independently of the mode theory, obtaining it for many different logics at once. Further, we give a general equational theory on derivations / terms that, in addition to the usual $\beta\eta$-rules, characterizes when two derivations differ only by the placement of structural rules. Additionally, we give an equivalent semantic presentation of these ideas, in which a mode theory corresponds to a 2-dimensional cartesian multicategory, the framework corresponds to another such multicategory with a functor to the mode theory, and the logical connectives make this into a bifibration. Finally, we show how the framework can be used both to encode existing existing logics / type theories and to design new ones.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases type theory, modal logic, substructural logic, homotopy type theory

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.25

1 Introduction

In ordinary intuitionistic logic or λ-calculus, assumptions or variables can go unused (weakening), be used in any order (exchange), be used more than once (contraction), and be used in any position in a term. *Substructural* logics, such as linear logic, ordered logic, relevant logic, and affine logic, omit some of these structural properties of weakening, exchange, and contraction, while *modal logics* place restrictions on where variables may be used – e.g. a formula $\Box\, C$ can only be proved using assumptions of $\Box\, A$, while an assumption of $\Diamond\, A$

* This material is based on research sponsored by The United States Air Force Research Laboratory under agreement number FA9550-15-1-0053 and FA9550-16-1-0292. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the United States Air Force Research Laboratory, the U.S. Government, or Carnegie Mellon University.

can only be used when the conclusion is $\Diamond C$. Substructural and modal logics have had many applications to both functional and logic programming, modeling concepts such state, staging, distribution, and concurrency. They are also used as *internal languages* of categories, where one uses an appropriate logical language to do constructions "inside" a particular mathematical setting, which often results in shorter statements than working "externally". For example, to define a function externally in domains, one must first define the underlying set-theoretic function, and then prove that it is continuous; when using untyped λ-calculus as an internal language of domains, one writes what looks like only the function part, and continuity follows from a general theorem about the language itself. Substructural logics extend this idea to various forms of monoidal categories, while modal logics describe monads and comonads. Recent work [30, 31] proposed using modal operators to add a notion of *cohesion* to homotopy type theory/univalent foundations [33, 32]. Without going into the precise details, the idea is to add a triple $\int \dashv \flat \dashv \sharp$ of type operators, where for example $\sharp$ and $\int$ are monads (like a modal possibility $\Diamond$ or $\bigcirc$), $\flat$ is a comonad (like a modal necessity $\Box$), and there is an adjunction structure between them ($\flat A \to B$ is the same as $A \to \sharp B$). This raised the question of how to best add modalities with these properties to type theory.

Because other similar applications rely on functors with different properties, we would like general tools for going from a semantic situation of interest to a well-behaved logic/type theory for it – e.g. one with cut admissibility / normalization and identity admissibility / η-expansion. In previous work [15], we considered the special case of a single-assumption logic, building most directly on adjoint logic [5, 6, 26]. Here we extend this previous work to the multi-assumption case. The resulting framework is quite general and covers many existing intuitionistic substructural and modal connectives: non-associative, ordered, linear, affine, relevant, and cartesian products and implications; combinations thereof such as bunched logic [21] and resource separation [3]; n-linear variables [24, 1, 17]; the comonadic $\Box$ and linear exponential ! and subexponentials [20, 10]; monadic $\Diamond$ and $\bigcirc$ modalities; and adjoint logic F and G [5, 6, 26], including the single-assumption 2-categorical version from our previous work [15]. A central syntactic result is that cut and identity are admissible for our framework itself, and this implies cut admissibility for any logic that can be described in the framework, including all of the above, as well as any new logics that one designs using it. When we view the derivations in the framework as terms in a type theory, this gives an immediate normalization (and η-expansion) result. Our focus here is on propositional, single-conclusioned substructural and modal logics, leaving extensions to quantifiers, multi-conclusioned logics, and dependent types to future work.

At a high level, the framework makes use of the fact that all of the above logics / type theories are a restriction on how variables can be used in ordinary structural/cartesian proofs. We express these restrictions using a first layer, which is a simple type theory for what we will call *modes* and *context descriptors*. The modes are just a collection of base types, which we write as p, q, r, while a context descriptor α is a first-order term built from variables and function symbols. The next layer is the main logic. Each proposition/type is assigned a mode, and the basic sequent is $x_1 : A_1, \ldots, x_n : A_n \vdash_\alpha C$, where if A_i has mode p_i, and C has mode q, then $x_1 : p_1, \ldots, x_n : p_n \vdash \alpha : q$. We use a sequent calculus to concisely describe cut-free derivations/normal forms, but everything can be translated to natural deduction. We write Γ for $x_1 : A_1, \ldots, x_n : A_n$, and Γ itself behaves like an ordinary structural/cartesian context, while the substructural and modal aspects are enforced by the *term* α, which constrains how the resources from Γ may be used. For example, in linear logic/ordered logic/BI, the context is usually taken to be a multiset/list/tree. We represent this by a pair of an ordinary structural context Γ, together with a term α that describes

the multiset or list or tree structure, labeled with variables from the ordinary context at the leaves. We pronounce $\Gamma \vdash_\alpha A$ as "Γ proves A {along,over,using} α".

For example, if we have a mode n, together with a context descriptor constant $x : \mathsf{n}, y : \mathsf{n} \vdash x \odot y : \mathsf{n}$, then an example sequent $x : A, y : B, z : C, w : D \vdash_{(y \odot x) \odot z} E$ should be read as saying that we must prove E using the resources y and x and z (but not w) according to the particular tree structure $(y \odot x) \odot z$. If we say nothing else, the framework will treat $\odot$ as describing a non-associative, linear, ordered context [14]: if we have a product-like type $A \odot B$ internalizing this context operation,[1] then we will *not* be able to prove associativity $((A \odot B) \odot C \dashv\vdash A \odot (B \odot C))$ or exchange $(A \odot B \vdash B \odot A)$ etc. To get from this basic structure to a linear or affine or relevant or cartesian system, we provide a way to add structural properties governing the context descriptor term α. We analyze structural properties as *equations*, or more generally *directed transformations*, on such terms. For example, to specify linear logic, we will add a unit element $1 : \mathsf{n}$ together with equations making $(\odot, 1)$ into a commutative monoid $(x \odot (y \odot z) = (x \odot y) \odot z$ and $x \odot 1 = x = 1 \odot x$ and $x \odot y = y \odot x)$ so that the context descriptors ignore associativity and order. To get BI, we add an additional commutative monoid $(\times, \top)$ (with weakening and contraction, as discussed below), so that a BI context tree $(x : A, y : B); (z : C, w : D)$ can be represented by the ordinary context $x : A, y : B, z : C, w : D$ with the term $(x \odot y) \times (z \odot w)$ describing the tree. Because the context descriptors are themselves ordinary structural/cartesian terms, the same variable can occur more than once or not at all. A descriptor such as $x \odot x$ captures the idea that we can use the *same* variable x twice, expressing n-linear types. Thus, we can express contraction for a particular context descriptor $\odot$ as a transformation $x \Rightarrow x \odot x$ (one use of x allows two). Weakening, on the other hand, is represented by a transformation $x \Rightarrow 1$, which is oriented to allow throwing away an allowed use of x, but not creating an allowed use from nothing. We refer to these as *structural transformations*, to evoke their use in representing the structural properties of object logics that are embedded in our framework. The main sequent $\Gamma \vdash_\alpha A$ respects the specified structural properties in the sense that when $\alpha = \beta$, we regard $\Gamma \vdash_\alpha A$ and $\Gamma \vdash_\beta A$ as the same sequent (so a derivation of one is a derivation of the other), while when $\alpha \Rightarrow \beta$, there will be an operation that takes a derivation of $\Gamma \vdash_\beta A$ to a derivation of $\Gamma \vdash_\alpha A$ – i.e. uses of transformations are explicitly marked in the term.

Modal logics will generally involve a mode theory with more than one mode. For example, a context descriptor $x : \mathsf{c} \vdash \mathsf{f}(x) : \mathsf{l}$ will generate an adjoint pair of functors between the two modes, as in the adjoint syntax for linear logic's ! [6] or other modal operators [26]. Using this, a context descriptor $\mathsf{f}(x) \odot y$ expresses permission to use x in a cartesian way and y in a linear way. Structural transformations are used to describe how these modal operators interact with each other and with the products, and for some systems [15] it is important that there can be more than one transformation between a given pair of context descriptors.

A guiding principle of the framework is a meta-level notion of *structurality over structurality*. For example, we always have *weakening over weakening*: if $\Gamma \vdash_\alpha A$ then $\Gamma, y : B \vdash_\alpha A$, where α itself is weakened with y. This does not prevent encodings of relevant logics: though we might weaken a derivation of $\Gamma \vdash_{x_1 \odot \ldots \odot x_n} A$ ("use x_1 through x_n") to a derivation of $\Gamma, y : B \vdash_{x_1 \odot \ldots \odot x_n} A$, the (weakened) context descriptor does not allow the use of y. Similarly, we have exchange over exchange and contraction over contraction. The *identity-over-identity* principle says that we should be able to prove A using exactly an assumption $x : A$ $(\Gamma, x : A \vdash_x A)$. The cut principle says that from $\Gamma, x : A \vdash_\beta B$ and $\Gamma \vdash_\alpha A$ we

[1] We overload binary operations to refer both to context descriptors and propositional connectives, relying on metavariables ($\alpha_1 \odot \alpha_2$ vs. $A_1 \odot A_2$) to distinguish them.

get $\Gamma \vdash_{\beta[\alpha/x]} B$ – the context descriptor for the result of the cut is the substitution of the context descriptor used to prove A into the one used to prove B. For example, together with weakening-over-weakening, this captures the usual cut principle of linear logic, which says that cutting $\Gamma, x : A \vdash B$ and $\Delta \vdash A$ yields $\Gamma, \Delta \vdash B$: if Γ binds $x_1, \ldots, x_n$ and Δ binds $y_1, \ldots, y_n$, then we will represent the two derivations to be cut together by sequents with $\beta = x_1 \odot \ldots \odot x_n \odot x$ and $\alpha = y_1 \odot \ldots \odot y_n$, so $\beta[\alpha/x] = x_1 \odot \ldots \odot x_n \odot y_1 \odot \ldots \odot y_n$ correctly deletes x and replaces it with the variables from Δ. In more subtle situations such as BI, the substitution will insert the resources used to prove the cut formula in the correct place in the tree. Our cut algorithm follows cut admissibility for structural (cartesian) intuitionistic logic [22], and applies the same cut reductions to all mode theories. In substructural situations, certain portions of this general algorithm are unnecessary, but not harmful. For example, in a setting without contraction, our algorithm will recursively cut into all premises of a rule, even though the variable can only occur in one premise – but the extra recursive cuts for variables that do not occur will leave the derivation unchanged, as desired.

The framework has two main logical connectives / type constructors. The first, $\mathsf{F}_\alpha(\Delta)$, generalizes the left-adjoint F of adjoint logic and the multiplicative products (e.g. $\otimes$ of linear logic). The second, $\mathsf{U}_{x.\alpha}(\Delta \mid A)$, generalizes the right-adjoint G/U of adjoint logic and implication (e.g. $A \multimap B$ in linear logic). Here Δ is a context of assumptions $x_i : A_i$, and trivializing the context descriptors (i.e. adding an equation $\alpha = \beta$ for all α and β) degenerates $\mathsf{F}_\alpha(\Delta)$ into the ordinary intuitionistic product $A_1 \times \ldots \times A_n$, while $\mathsf{U}_{x.\alpha}(\Delta \mid A)$ becomes $A_1 \to \ldots \to A_n \to A$. As one would expect, F is left-invertible and U is right-invertible. In linear logic terms, our F and U cover both the multiplicatives and exponentials; additives can be defined separately by the usual rules. Moreover, though F and U form an adjoint pair, the subset of derivations that use either F_α or U_α but not both (for a particular α) describe constructions on functors that do not have an adjoint. We discuss many examples of *logical adequacy* theorems, showing that a sequent can be proved in a standard sequent calculus for a logic iff its embedding using these connectives can be proved in the framework.

Being a very general theory, our framework treats the object-logic structural properties in a general but naïve way, allowing an arbitrary structural transformation to be applied at the non-invertible rules for F and U and at the leaves of a derivation. For specific embedded logics, there is often a more refined discipline that suffices – e.g. for cartesian logic, always contract all assumptions in all premises, and only weaken at the leaves. We view our framework as a tool for bridging the gap between an intended semantic situation (in the cohesion example mentioned, "a comonad and a monad which are themselves adjoint") and a proof theory: the framework gives *some* proof theory for the semantics, and the placement of structural rules can then be optimized purely in syntax. To support this mode of use, we give an equational theory on derivations/terms that identifies different placements of the same structural rules. This can be used to prove correctness of such optimizations not just at the level of provability, but also identity of derivations – which matters for our intended applications to internal languages. We discuss some preliminary work on *equational adequacy*, which extends the logical correspondence to isomorphisms of definitional-equality-classes of derivations.

Semantically, the logic corresponds to a functor between *2-dimensional cartesian multicategories* which is a fibration in various senses. Multicategories are a generalization of categories which allow more than one object in the domain of morphisms, and cartesianness means that the multiple domain objects are treated structurally. The 2-dimensionality supplies a notion of morphism between (multi)morphisms. A *mode theory* specifying context descriptors and structural properties is analyzed as a cartesian 2-multicategory, with the descriptors as 1-cells and the structural properties as 2-cells. The functor relates the sequent

judgement to the mode theory, specifying the mode of each proposition and the context descriptor of a sequent. The fibration conditions (similar to [12, 13]) give respect for the structural transformations and the presence of F and U types. We prove that the sequent calculus and the equational theory are sound and complete for this semantics: the syntax can be interpreted in any bifibration, and itself determines one. This semantics shows that an interesting class of type theories can be identified with a class of more mathematical objects, fibrations of cartesian 2-multicategories, thus providing some progress towards characterizing substructural and modal type theories in mathematical terms.

In Section 2, we present the syntax of the framework. In Section 3, we discuss how a number of logics are represented. In Section 4, we give the $\beta\eta$-equational theory on derivations. In Section 5, we discuss the framework's categorical semantics. Proofs (of cut and identity admissibility, soundness and completeness of the semantics, and adequacy of encodings) and additional examples (subexponentials, modal S4 $\Box$, and strong/$\Box$-strong monads) are available in an extended version of this paper [16].

2 Sequent Calculus

2.1 Mode Theories

The first layer of our framework is a type theory whose types we will call *modes*, and whose terms we will call *context descriptors* or *mode morphisms*. To a first approximation, context descriptors are multi-sorted first-order terms with equality and "less than or equal to" relations. The only modes are atomic/base types p. A term is either a variable (bound in a context ψ) or a typed n-ary constant (function symbol) c applied to terms of the appropriate types. Two terms may be equal, written $\alpha \equiv \beta$, or α may be stronger than β, written $\alpha \Rightarrow \beta$.

This is formalized in the notion of signature, or *mode theory*, defined in Figure 1. The judgement $\Sigma\,\mathsf{sig}$ means that Σ is a well-formed signature. The top line says that a signature is either empty, or a signature extended with a new mode declaration, or a signature extended with a typed constant/function symbol, all of whose modes are declared previously in the signature. The notation $p_1, \ldots, p_n \to q$ is not itself a mode, but notation for declaring a function symbol in the signature (it cannot occur on the right-hand side of a typing judgement). For example, the type and term constructors for a monoid $(\odot, 1)$ are represented by a signature $\mathsf{p}\,\mathsf{mode}, \odot : (\mathsf{p}, \mathsf{p} \to \mathsf{p}), 1 : (\to \mathsf{p})$.

We elide the rules for the judgement $\vdash_\Sigma \psi\,\mathsf{ctx}$, which simply says that each mode used in the context of variable declarations ψ is declared in Σ. The judgement $\psi \vdash_\Sigma \alpha : p$ defines well-typedness of context descriptor terms, which are either a variable declared in the context, or a constant declared in the signature applied to arguments of the correct types. The judgement $\psi \vdash_\Sigma \gamma : \psi'$ defines a substitution as a tuple of terms in the standard way. The context ψ in these judgements enjoys the cartesian structural properties (associativity, unit, weakening, exchange, contraction). Simultaneous substitution into terms and substitutions is defined as usual (e.g. $x[\gamma, \alpha/x] := \alpha$ and $\mathsf{c}(\vec{\alpha_i})[\gamma] := \mathsf{c}(\alpha_i[\vec{\gamma}])$).

Returning to the top of the figure, the final two rules of the judgement $\Sigma\,\mathsf{sig}$ permit two additional forms of signature declaration. The first of these extends a signature with an equational axiom between two terms α and α' that have the same mode p, in the same context ψ, relative to the prior signature Σ. These equational axioms will be used to encode reversible object language structural properties, such as associativity, commutativity, and unit laws. For example, to specify the right unit law for the above monoid $(\odot, 1)$, we add an axiom $(x \odot 1 \equiv x : (x : \mathsf{p}) \to \mathsf{p})$ to the signature, which can be read as "$x \odot 1$ is equal to x as a morphism from $(x : \mathsf{p})$ to p". The judgement $\psi \vdash_\Sigma \alpha \equiv \alpha' : p$ (omitted from the figure;

Signatures $\Sigma\,\mathsf{sig}$

$$\frac{}{\cdot\,\mathsf{sig}} \qquad \frac{\Sigma\,\mathsf{sig}}{(\Sigma, p\,\mathsf{mode})\,\mathsf{sig}} \qquad \frac{\Sigma\,\mathsf{sig} \quad (p_1\,\mathsf{mode}, \ldots, p_n\,\mathsf{mode}, q\,\mathsf{mode}) \in \Sigma}{(\Sigma, c :\ p_1, \ldots, p_n \to q)\,\mathsf{sig}}$$

$$\frac{\Sigma\,\mathsf{sig} \quad \vdash_\Sigma \psi\,\mathsf{ctx} \quad p\,\mathsf{mode} \in \Sigma \quad \psi \vdash_\Sigma \alpha : p \quad \psi \vdash_\Sigma \alpha' : p}{(\Sigma, (\alpha \equiv \alpha' : \psi \to p))\,\mathsf{sig}}$$

$$\frac{\Sigma\,\mathsf{sig} \quad \vdash_\Sigma \psi\,\mathsf{ctx} \quad p\,\mathsf{mode} \in \Sigma \quad \psi \vdash_\Sigma \alpha : p \quad \psi \vdash_\Sigma \alpha' : p}{(\Sigma, (\alpha \Rightarrow \alpha' : \psi \to p))\,\mathsf{sig}}$$

Context descriptors $\psi \vdash_\Sigma \alpha : p$, where $\vdash_\Sigma \psi\,\mathsf{ctx}$ and $p\,\mathsf{mode} \in \Sigma$

$$\frac{x : p \in \psi}{\psi \vdash_\Sigma x : p} \qquad \frac{(\mathsf{c} : p_1, \ldots, p_n \to q) \in \Sigma \quad \psi \vdash_\Sigma \alpha_i : p_i}{\psi \vdash_\Sigma \mathsf{c}(\alpha_1, \ldots, \alpha_n) : q}$$

Mode Substitutions $\psi \vdash_\Sigma \gamma : \psi'$, where $\vdash_\Sigma \psi\,\mathsf{ctx}$ and $\vdash_\Sigma \psi'\,\mathsf{ctx}$

$$\frac{}{\psi \vdash_\Sigma \cdot : \cdot} \qquad \frac{\psi \vdash_\Sigma \gamma : \psi' \quad \psi \vdash_\Sigma \alpha : p}{\psi \vdash_\Sigma \gamma, \alpha/x : \psi', x : p}$$

Structural transformations $\psi \vdash_\Sigma \alpha \Rightarrow_p \alpha'$, where $\psi \vdash_\Sigma \alpha : p$ and $\psi \vdash_\Sigma \alpha' : p$

$$\frac{}{\psi \vdash_\Sigma \alpha \Rightarrow_p \alpha} \qquad \frac{\psi \vdash_\Sigma \alpha_1 \Rightarrow_p \alpha_2 \quad \psi \vdash_\Sigma \alpha_2 \Rightarrow_p \alpha_3}{\psi \vdash_\Sigma \alpha_1 \Rightarrow_p \alpha_3}$$

$$\frac{\psi, x : p, \psi' \vdash_\Sigma \beta \Rightarrow_q \beta' \quad \psi, \psi' \vdash_\Sigma \alpha \Rightarrow_p \alpha'}{\psi, \psi' \vdash_\Sigma \beta[\alpha/x] \Rightarrow_q \beta'[\alpha'/x]} \qquad \frac{(\alpha \Rightarrow \alpha' : \psi \to p) \in \Sigma}{\psi \vdash_\Sigma \alpha \Rightarrow_p \alpha'}$$

Figure 1 Syntax for mode theories.

the rules are the same as for $\Rightarrow$ plus symmetry) is the least congruence closed under these axioms.

The second of these extends a signature with a directed structural transformation axiom between two terms α and α' that have the same mode p, in the same context ψ, relative to the prior signature Σ. As discussed above, these structural transformations will be used to represent object language structural properties such as weakening and contraction that are not invertible. The judgement $\psi \vdash_\Sigma \alpha \Rightarrow_p \alpha'$ defines these transformations: it is the least precongruence (preorder compatible with the term formers) closed under the axioms specified in the signature Σ. For example, to say that the above monoid $(\odot, 1)$ is affine, we add in Σ a transformation axiom $(x \Rightarrow 1 : (x : \mathsf{p}) \to \mathsf{p})$.

Because context descriptors α and their equality $\alpha_1 \equiv \alpha_2$ are defined prior to the subsequent judgements, we suppress this equality by using α to refer to a term-modulo-$\equiv$– that is, we assume a metatheory with quotient sets/types, and use meta-level equality for object-level equality [2]. For example, because the judgement $\psi \vdash \alpha \Rightarrow_p \beta$ is indexed by equivalence classes of context descriptions, the reflexivity rule above implicitly means $\alpha \equiv \beta$ implies $\alpha \Rightarrow \beta$. In examples, we will notate a signature declaration introducing a term constant/function symbol by showing the function symbol applied to variables, rather than

$$\boxed{\text{Types } A, B, C \quad A \ \mathsf{type}_p}$$

$$\frac{}{P \ \mathsf{type}_p} \qquad \frac{\psi \vdash \alpha : q \quad \Delta \ \mathsf{ctx}_\psi}{\mathsf{F}_\alpha(\Delta) \ \mathsf{type}_q} \qquad \frac{\psi, x : q \vdash \alpha : p \quad \Delta \ \mathsf{ctx}_\psi \quad A \ \mathsf{type}_p}{\mathsf{U}_{x.\alpha}(\Delta \mid A) \ \mathsf{type}_q}$$

$$\boxed{\text{Contexts } \Gamma, \Delta \quad \Gamma \ \mathsf{ctx}_\psi}$$

$$\frac{}{\cdot \ \mathsf{ctx}.} \qquad \frac{\Gamma \ \mathsf{ctx}_\psi \quad A \ \mathsf{type}_p}{\Gamma, x : A \ \mathsf{ctx}_{\psi, x:p}}$$

$$\boxed{\Gamma \vdash_\alpha A \text{ where } \Gamma \ \mathsf{ctx}_\psi \text{ and } A \ \mathsf{type}_q \text{ and } \psi \vdash \alpha : q}$$

$$\frac{\Gamma, \Gamma', \Delta \vdash_{\beta[\alpha/x]} C}{\Gamma, x : \mathsf{F}_\alpha(\Delta), \Gamma' \vdash_\beta C} \ \mathsf{FL} \qquad \frac{\beta \Rightarrow \alpha[\gamma] \quad \Gamma \vdash_\gamma \Delta}{\Gamma \vdash_\beta \mathsf{F}_\alpha(\Delta)} \ \mathsf{FR}$$

$$\frac{x : \mathsf{U}_{x.\alpha}(\Delta \mid A) \in \Gamma \quad \beta \Rightarrow \beta'[\alpha[\gamma]/z] \quad \Gamma \vdash_\gamma \Delta \quad \Gamma, z : A \vdash_{\beta'} C}{\Gamma \vdash_\beta C} \ \mathsf{UL}$$

$$\frac{\Gamma, \Delta \vdash_{\alpha[\beta/x]} A}{\Gamma \vdash_\beta \mathsf{U}_{x.\alpha}(\Delta \mid A)} \ \mathsf{UR} \qquad \frac{x : P \in \Gamma \quad \beta \Rightarrow x}{\Gamma \vdash_\beta P} \ \mathsf{v}$$

$$\boxed{\Gamma \vdash_\gamma \Delta \text{ where } \Gamma \ \mathsf{ctx}_\psi \text{ and } \Delta \ \mathsf{ctx}_{\psi'} \text{ and } \psi \vdash \gamma : \psi'}$$

$$\frac{}{\Gamma \vdash_\cdot \cdot} \ \cdot \qquad \frac{\Gamma \vdash_\gamma \Delta \quad \Gamma \vdash_\alpha A}{\Gamma \vdash_{\gamma, \alpha/x} \Delta, x : A} \ \_,\_$$

Figure 2 Sequent Calculus.

writing the formal $\mathsf{c} : p_1, \ldots, p_n \to q$. For example, we write $x : \mathsf{p}, y : \mathsf{p} \vdash x \odot y : \mathsf{p}$ for $\odot : \mathsf{p}, \mathsf{p} \to \mathsf{p}$. We also suppress the signature Σ.

2.2 Sequent Calculus Rules

For a fixed mode theory Σ, we define a second layer of judgements in Figure 2. The first judgement assigns each proposition/type A a mode p. Encodings of non-modal logics will generally only make use of one mode, while modal logics use different modes to represent different notions of truth, such as the linear and cartesian categories in the adjoint decomposition of linear logic [5, 6] and the true/valid/lax judgements in modal logic [23]. The next judgement assigns each context Γ a mode context ψ. Formally, we think of contexts as ordered: we do not regard $x : A, y : B$ and $y : B, x : A$ as the same context, though we will have an admissible exchange rule that passes between derivations in one and the other.

The sequent judgement $\Gamma \vdash_\alpha A$ relates a context $\Gamma \ \mathsf{ctx}_\psi$ and a type $A \ \mathsf{type}_p$ and context descriptor $\psi \vdash \alpha : p$, while the substitution judgement $\Gamma \vdash_\gamma \Delta$ relates $\Gamma \ \mathsf{ctx}_\psi$ and $\Delta \ \mathsf{ctx}_{\psi'}$ and $\psi \vdash \gamma : \psi'$. Because $\Gamma \ \mathsf{ctx}_\psi$ means that each variable in Γ is in ψ, where $x : A_i \in \Gamma$ implies $x : p_i$ in ψ with $A_i \ \mathsf{type}_{p_i}$, we think of Γ as binding variable names both in α and for use in the derivation.

We now explain the rules for the sequent calculus; the reader may wish to refer to the examples in Section 3 in parallel with this abstract description. We assume atomic propositions P are given a specified mode p, and state identity as a primitive rule only for them with the v rule. This says that $\Gamma, x : P \vdash_x P$, and additionally composes with a structural transformation $\beta \Rightarrow x$. Using a structural property at a leaf of a derivation is common in e.g. affine logic, where the derivation of $\beta \Rightarrow x$ would use weakening to forget any additional resources besides x.

Next, we consider the $\mathsf{F}_\alpha(\Delta)$ type, which "internalizes" the context operation α as a type/proposition. Syntactically, we view the context $\Delta = x_1 : A_1, \ldots, x_n : A_n$ where A_i type_{p_i} as binding the variables $x_i : p_i$ in α, so for example $\mathsf{F}_\alpha(x : A, y : B)$ and $\mathsf{F}_{\alpha[x \leftrightarrow x']}(x' : A, y : B)$ are α-equivalent types (in de Bruijn form we would write $\mathsf{F}_\alpha(A_1, \ldots, A_n)$ and use indices in α). The type formation rule says that F moves covariantly along a mode morphism α, representing a "product" (in a loose sense) of the types in Δ structured according to the context descriptor α. A typical binary instance of F is a multiplicative product ($A \otimes B$ in linear logic), which, given a binary context descriptor $\odot$ as in the introduction, is written $\mathsf{F}_{x \odot y}(x : A, y : B)$. A typical nullary instance is a unit (1 in linear logic), written $\mathsf{F}_1()$. A typical unary instance is the F connective of adjoint logic, which for a unary context descriptor constant $\mathsf{f} : \mathsf{p} \to \mathsf{q}$ is written $\mathsf{F}_{\mathsf{f}(x)}(x : A)$. We sometimes write $\mathsf{F}_\mathsf{f}(A)$ in this case, eliding the variable name, and similarly for a unary U.

The rules for our F connective capture a pattern common to all of these examples. The left FL rule says that $\mathsf{F}_\alpha(\Delta)$ "decays" into Δ, but structuring the uses of resources in Δ with α by the substitution $\beta[\alpha/x]$. We assume that Δ is α-renamed to avoid collision with Γ (the proof term here is a "split" that binds variables for each position in Δ). The placement of Δ at the right of the context is arbitrary (because we have exchange-over-exchange), but we follow the convention that new variables go on the right to emphasize that Γ behaves mostly as in ordinary cartesian logic. The right FR rule says that you must rewrite (using structural transformations) the context descriptor to have an α at the outside, with a mode substitution γ that divides the existing resources up between the positions in Δ, and then prove each formula in Δ using the specified resources. We leave the typing of γ implicit, though there is officially a requirement $\psi \vdash \gamma : \psi'$ where Γ ctx_ψ and Δ $\mathsf{ctx}_{\psi'}$, as required for the second premise to be a well-formed sequent. Another way to understand this rule is to begin with the "axiomatic FR" instance $\mathsf{FR}^* :: \Delta \vdash_\alpha \mathsf{F}_\alpha(\Delta)$ which says that there is a map from Δ to $\mathsf{F}_\alpha(\Delta)$ along α. Then, in the same way that a typical right rule for coproducts builds a precomposition into an "axiomatic injection" such as $\mathsf{inl} :: A \vdash A + B$, the FR rule builds a precomposition with $\Gamma \vdash_\gamma \Delta$ and then an application of a structural rule $\beta \Rightarrow \alpha[\gamma]$ into the "axiomatic" version, in order to make cut and respect for transformations admissible.

Next, we turn to $\mathsf{U}_{x.\alpha}(\Delta \mid A)$. As a first approximation, if we ignore the context descriptors and structural properties, $\mathsf{U}_\_(\Delta \mid A)$ behaves like $\Delta \to A$, and the UL and UR rules are an annotation of the usual structural/cartesian rules for implication. In a formula $\mathsf{U}_{x.\alpha}(\Delta \mid A)$, the context descriptor α has access to the variables from Δ as well as an extra variable x, whose mode is the same as the *overall mode of* $\mathsf{U}_{x.\alpha}(\Delta \mid A)$, while the mode of A itself is the mode of the conclusion of α – in terms of typing, U is contravariant where F is covariant. It is helpful to think of x as standing for the context that will be used to prove $\mathsf{U}_{x.\alpha}(\Delta \mid A)$. For example, a typical function type $A \multimap B$ is represented by $\mathsf{U}_{x.x \otimes y}(y : A \mid B)$, which says to extend the "current context" x with a resource y. In UR, the context descriptor β being used to prove the U is substituted *for* x in α (dual to FL, which substituted α into β). The "axiomatic" UL instance $\mathsf{UL}^* :: \Delta, x : \mathsf{U}_{x.\alpha}(\Delta \mid A) \vdash_\alpha A$ says that $\mathsf{U}_{x.\alpha}(\Delta \mid A)$ together with Δ has a map to A along α. (The bound x in $x.\alpha$ subscript is tacitly renamed

to match the name of the assumption in the context, in the same way that the typing rule for $\lambda x.e : \Pi x : A.B$ requires coordination between two variables in different scopes). The full rule builds in precomposition with $\Gamma \vdash_\gamma \Delta$, postcomposition with $\Gamma, z : A \vdash_{\beta'} C$, and precomposition with $\beta \Rightarrow \beta'[\alpha[\gamma]/z]$.

Finally, the rules for substitutions are pointwise. In examples, we will write the components of a substitution directly as multiple premises of FR and UL, rather than packaging them with $\_,\_$ and $\cdot$.

For additives, the context descriptor is not modified; for example, a coproduct/disjunction $A_p + B_p$ type_p for a mode p is given by the following rules:

$$\frac{\Gamma \vdash_\alpha A}{\Gamma \vdash_\alpha A + B} \qquad \frac{\Gamma \vdash_\alpha B}{\Gamma \vdash_\alpha A + B} \qquad \frac{\Gamma, \Gamma', y : A \vdash_{\beta[y/x]} C \quad \Gamma, \Gamma', z : B \vdash_{\beta[z/x]} C}{\Gamma, x : A + B, \Gamma' \vdash_\beta C}$$

Our framework enjoys the following admissible structural rules:

▶ **Theorem 2.1** (Admissibility of cut, identity, structurality-over-structurality, and respect for 2-cells)**.** *The following rules are admissible:*

$$\frac{\Gamma, x : A \vdash_\beta B \quad \Gamma \vdash_\alpha A}{\Gamma \vdash_{\beta[\alpha/x]} B} \qquad \frac{}{\Gamma, x : A \vdash_x A} \qquad \frac{\Gamma \vdash_\alpha C}{\Gamma, y : A \vdash_\alpha C}$$

$$\frac{\Gamma, x : A, y : B \vdash_\alpha C}{\Gamma, y : B, x : A \vdash_\alpha C} \qquad \frac{\alpha \Rightarrow \beta \quad \Gamma \vdash_\beta A}{\Gamma \vdash_\alpha A}$$

The following general constructions can be helpful for understanding how the types behave. We write $A \vdash B$ for $x : A \vdash_x B$. The three "fusion" rules on the left (which are type isomorphisms, not just interprovabilities) relate F and U. Special cases include: $A \times (B \times C)$ is isomorphic to a primitive triple product $\{x : A, y : B, z : C\}$; currying; and associativity of n-ary functions $(A_1, \ldots, A_n \to (B_1, \ldots, B_m \to C)$ is isomorphic to $A_1, \ldots, A_n, B_1, \ldots, B_m \to C)$. Second, the types respect a transformation covariantly for F and contravariantly for U.

▶ **Theorem 2.2** (Fusion and Respect Laws)**.**

$$\begin{array}{rcl} \mathsf{F}_\alpha(\Delta, x : \mathsf{F}_\beta(\Delta'), \Delta'') & \dashv\vdash & \mathsf{F}_{\alpha[\beta/x]}(\Delta, \Delta', \Delta'') \\ \mathsf{U}_{x.\alpha}(\Delta, y : \mathsf{F}_\beta(\Delta'), \Delta'' \mid A) & \dashv\vdash & \mathsf{U}_{x.\alpha[\beta/y]}(\Delta, \Delta', \Delta'' \mid A) \\ \mathsf{U}_{x.\alpha}(\Delta \mid \mathsf{U}_{y.\beta}(\Delta' \mid A)) & \dashv\vdash & \mathsf{U}_{x.\beta[\alpha/y]}(\Delta, \Delta' \mid A) \end{array}$$

$$\begin{array}{rcll} \mathsf{F}_\alpha(\Delta) & \vdash & \mathsf{F}_\beta(\Delta) & \textit{if } \alpha \Rightarrow \beta \\ \mathsf{U}_{x.\beta}(\Delta \mid A) & \vdash & \mathsf{U}_{x.\alpha}(\Delta \mid A) & \textit{if } \alpha \Rightarrow \beta \end{array}$$

3 Examples

3.1 Products and Implications

First, we show how to encode substructural products and implications with various structural properties. A mode theory with one mode m and a constant $x : \mathsf{m}, y : \mathsf{m} \vdash x \odot y : \mathsf{m}$ specifies a completely astructural context (no weakening, exchange, contraction, associativity), as in non-associative Lambek calculus [14]. To pass to *ordered logic* (associativity and unit laws but none of exchange, weakening, and contraction), we add a constant $1 : \mathsf{m}$ and equational axioms $x \odot (y \odot z) \equiv (x \odot y) \odot z$ and $x \odot 1 \equiv x \equiv 1 \odot x$ – i.e. $(\odot, 1)$ is a monoid. To

get linear logic, we additionally add commutativity $x \odot y \equiv y \odot x$. As a first example of using the sequent calculus, we show how commutativity of $\odot$ in the mode theory for linear logic generates commutativity of the corresponding $A \otimes B$ type, which is represented by $\mathsf{F}_{x \odot y}(x : A, y : B)$:

$$\dfrac{\dfrac{x \odot y \Rightarrow (z \odot w)[y/z, x/w] \quad x : A, y : B \vdash_y B \quad x : A, y : B \vdash_x A}{x : A, y : B \vdash_{x \odot y} \mathsf{F}_{z \odot w}(z : B, w : A)}\ \mathsf{FR}}{q : \mathsf{F}_{x \odot y}(x : A, y : B) \vdash_q \mathsf{F}_{z \odot w}(z : B, w : A)}\ \mathsf{FL}$$

First, we use FL to split the product type on the left up, obtaining permission to use its pieces by substituting $(x \odot y)$ for the variable q we began with. Next, to use FR, we must transform the current context descriptor $x \odot y$ into a substitution instance of the one from the type $z \odot w$ – dividing our resources in the form dictated by the type. We take $y/z, x/w$, which requires a transformation $x \odot y \Rightarrow y \odot x$, which is given by reflexivity because of the commutativity axiom in the mode theory. Then we can prove each of A and B by identity, because we have the correct resources in each branch. In the mode theory for ordered logic, without commutativity, the only possible division is $x/z, y/w$, and with permission only to use x the first premise and y in the second, the derivation fails.

Returning to the mode theory of a non-symmetric $\odot$, we show how the two implications/residuations of ordered logic are modeled by U-types; the expected rules are

$$\dfrac{\Gamma, A \vdash^\circ B}{\Gamma \vdash^\circ A \rightharpoonup B} \qquad \dfrac{\Delta \vdash^\circ A \quad \Gamma, B, \Gamma' \vdash^\circ C}{\Gamma, A \rightharpoonup B, \Delta, \Gamma' \vdash^\circ C} \qquad \dfrac{A, \Gamma \vdash^\circ B}{\Gamma \vdash^\circ A \leftharpoonup B} \qquad \dfrac{\Delta \vdash^\circ A \quad \Gamma, B, \Gamma' \vdash^\circ C}{\Gamma, \Delta, A \leftharpoonup B, \Gamma' \vdash^\circ C}$$

We represent these by the U-types $A \rightharpoonup B := \mathsf{U}_{c.c \odot x}(x : A \mid B)$ and $A \leftharpoonup B := \mathsf{U}_{c.x \odot c}(x : A \mid B)$. The UL and UR rules specialize as follows:

$$\dfrac{\Gamma, x : A \vdash_{\beta \odot x} B}{\Gamma \vdash_\beta \mathsf{U}_{c.c \odot x}(x : A \mid B)} \qquad \dfrac{\begin{array}{l} c : \mathsf{U}_{c.c \odot x}(x : A \mid B) \in \Gamma \\ \beta \Rightarrow \beta'[c \odot \alpha / z] \\ \Gamma \vdash_\alpha A \\ \Gamma, z : A \vdash_{\beta'} C \end{array}}{\Gamma \vdash_\beta C}$$

$$\dfrac{\Gamma, x : A \vdash_{x \odot \beta} B}{\Gamma \vdash_\beta \mathsf{U}_{c.x \odot c}(x : A \mid B)} \qquad \dfrac{\begin{array}{l} c : \mathsf{U}_{c.x \odot c}(x : A \mid B) \in \Gamma \\ \beta \Rightarrow \beta'[\alpha \odot c / z] \\ \Gamma \vdash_\alpha A \\ \Gamma, z : A \vdash_{\beta'} C \end{array}}{\Gamma \vdash_\beta C}$$

The UR instances put x on the left or right of the current context descriptor β, by the substitution β/c in UR. Consider the left rule for $\rightharpoonup / \mathsf{U}_{c.c \odot x}(x : A \mid B)$, and suppose that the β in the conclusion is of the form $x_1 \odot \ldots c \ldots \odot x_n$ for distinct variables x_i. Because the only structural transformations are the associativity and unit equations, the transformation must reassociate β as $\beta_1 \odot (c \odot \alpha) \odot \beta_2$, with $\beta' = \beta_1 \odot z \odot \beta_2$, for some β_1 and β_2. Here α plays the role of Δ in the ordered logic rule – the resources used to prove A, which occur to the right of the implication being eliminated. Reading the substitution backwards, the resources β' used for the continuation are "β with $c \odot \alpha$ replaced by the result of the implication," as desired. While c and any variables used in α are still in Γ, permission to use them has been removed from β' – and there is no way to restore such permissions in this mode theory. The rule for $\leftharpoonup$ is the same, but with α on the opposite side of c. For the linear logic mode theory, $\mathsf{U}_{c.c \odot x}(x : A \mid B)$ and $\mathsf{U}_{c.x \odot c}(x : A \mid B)$ are equal types (because commutativity is an equation, and types are parametrized by equivalence-classes of context descriptors), and both represent $A \multimap B$.

Weakening (affine logic) is modeled by adding a directed structural transformation $\mathsf{w} :: x \Rightarrow 1$, while contraction (relevant logic) is modeled by $\mathsf{c} :: x \Rightarrow x \odot x$. As an example

use of weakening, we can show $A \odot B \vdash A$ (formally $z : \mathsf{F}_{x \odot y}(x : A, y : B) \vdash_z A$); and as an example of contraction we can show $A \vdash A \odot A$ (formally $z : A \vdash_z \mathsf{F}_{x \odot y}(x : A, y : A)$):

$$\dfrac{\dfrac{\dfrac{\mathsf{w} :: y \Rightarrow 1}{x \odot y \Rightarrow x \odot 1 \equiv x} \quad \overline{x : A, y : B \vdash_x A}}{x : A, y : B \vdash_{x \odot y} A}}{z : \mathsf{F}_{x \odot y}(x : A, y : B) \vdash_z A}\ \mathsf{FL} \qquad \dfrac{\mathsf{c} :: z \Rightarrow (x \odot y)[z/x, z/y] \quad \overline{z : A \vdash_z A}}{z : A \vdash_z \mathsf{F}_{x \odot y}(x : A, y : A)}\ \mathsf{FR}$$

If we have both $\mathsf{w} :: x \Rightarrow 1$ and $\mathsf{c} :: x \Rightarrow x \odot x$ (with some equations relating them), then $x \odot y$ is a cartesian product in the mode theory, and consequently the type $\mathsf{F}_{x \odot y}(x : A, y : B)$ will behave like a cartesian product type $A \times B$, and $\mathsf{U}_{c.c \odot x}(x : A \mid B)$ like the usual structural $A \to B$. We refer to this mode theory as a *cartesian monoid* and write $(\times, \top)$ for it.

These encodings are adequate in the following sense:

▶ **Theorem 3.1** (Logical Adequacy for Products and Implications). *Write A^* for the encoding of a type as above and extend this pointwise to contexts Γ^*. Further, define $\overline{x_1 : A_1, \ldots, x_n : A_n} = x_1 \odot \ldots \odot x_n$. Then $\Gamma \vdash A$ in the standard sequent calculus iff $\Gamma^* \vdash_{\overline{\Gamma}} A^*$.*

Proof. Proofs for ordered logic (products), affine logic, and cartesian logic are in the extended version. Encoding an object-language derivation is straightforward, because the mode theory is chosen to make each rule derivable. The back-translation from the framework relies on cut-freeness (so that we only need to back-translate normal forms), and a lemma that, for these mode theories, left-rules on variables that are in the framework context Γ but do not occur in the context descriptor α can be strengthened away. ◀

This approach extends to contexts with more than one type of tree node, as in bunched implication [21], which has two context-forming operations Γ, Γ' and $\Gamma; \Gamma'$, along with corresponding products and implications. Both are associative, unital, and commutative, but ; has weakening and contraction while , does not. A context is represented by a tree such as $(x : A, y : B); (z : C, w : D)$ (considered modulo the laws), and the notation $\Gamma[\Delta]$ is used to refer to a tree with a hole $\Gamma[-]$ that has Δ as a subtree at the hole. In sequent calculus style, the rules for the product and implication corresponding to , are

$$\dfrac{\Gamma[A, B] \vdash C}{\Gamma[A * B] \vdash C} \quad \dfrac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A * B} \quad \dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \mathrel{-\!\!*} B} \quad \dfrac{\Delta \vdash A \quad \Gamma[B] \vdash C}{\Gamma[A \mathrel{-\!\!*} B, \Delta] \vdash C}$$

We model BI by a mode m with both a commutative monoid $(*, I)$ and a cartesian monoid $(\times, \top)$. We define the BI products and implications using the monoids as above: $A * B := \mathsf{F}_{x * y}(x : A, y : B)$ and $A \times B := \mathsf{F}_{x \times y}(x : A, y : B)$ and $A \mathrel{-\!\!*} B := \mathsf{U}_{c.c * x}(x : A \mid B)$ and $A \to B := \mathsf{U}_{c.c \times x}(x : A \mid B)$. A context descriptor such as $(x \times y) * (z \times w)$ captures the "bunched" structure of a BI context, and substitution for a variable models the hole-filling operation $\Gamma[\Delta]$. The derived left rules for $*$ and $\mathrel{-\!\!*}$ are

$$\dfrac{\Gamma, \Gamma', x : A, y : B \vdash_{\beta[x * y/z]} C}{\Gamma, z : A * B, \Gamma' \vdash_\beta C} \qquad \dfrac{c : A \mathrel{-\!\!*} B \in \Gamma \quad \beta \Rightarrow \beta'[c * \alpha/z] \quad \Gamma \vdash_\alpha A \quad \Gamma, z : B \vdash_{\beta'} C}{\Gamma \vdash_\beta C}$$

The rule for $*$ (and similarly $\times$) acts on a leaf z and replaces the leaf where z occurs in the tree β with the correct bunch $x * y$. The left rule for $\mathrel{-\!\!*}$ (and similarly for $\to$) isolates a subtree containing the implication c and resources $*$'ed with it, uses those resources to prove A, and then replaces the subtree with the variable z standing for the result of the implication.

3.2 Multi-use variables

An n-use variable [24, 1, 17] is a variable that is used "exactly n times" (modulo additives), as expressed by the following sequent calculus rules for n-use functions

$$\frac{}{0 \cdot \Gamma, x :^1 P \vdash P} \qquad \frac{\Gamma, x :^n A \vdash B}{\Gamma \vdash A \to^n B} \qquad \frac{\Delta \vdash A \quad \Gamma, z :^k B \vdash C}{\Gamma + f :^k A \to^n B + (nk \cdot \Delta) \vdash C}$$

where $\Gamma + \Delta$ acts pointwise by $x :^n A + x :^m A = x :^{n+m} A$ and $n \cdot \Delta$ acts pointwise by $n \cdot x^m A = x :^{nm} A$. In the left rule, Γ and Δ have the same underlying variables and types (but potentially different counts), and $f :^k A \to^n B$ abbreviates a context with the same variables and types but 0's for all counts besides f's. The left rule says that if you spend k "uses" of a function that takes n uses of an argument, then you need nk uses of whatever you use to construct the argument, in order to get k uses of the result.

We can model this in the mode theory of a commutative monoid by using context descriptors that are themselves non-linear: we define $A \to^n B := \mathsf{U}_{c.c\odot(x^n)}(x : A \mid B)$ where $x^n := x \odot x \odot \ldots \odot x$ (n times). This has the following instances of UL and UR:

$$\frac{\Gamma, x : A \vdash_{\beta \odot x^n} B}{\Gamma \vdash_\beta A \to^n B} \qquad \frac{f : \mathsf{U}_{f.f\odot x^n}(x : A \mid B) \in \Gamma \quad \beta \Rightarrow \beta'[f \odot (\alpha)^n/z] \quad \Gamma \vdash_\alpha A \quad \Gamma, z : B \vdash_{\beta'} C}{\Gamma \vdash_\beta C}$$

In the left rule, β' must be equal to some term $\beta'' \odot z^k$ for some k and β'' not mentioning z (for this mode theory, any term is a polynomial of variables), and the only structural transformations are the commutative monoid equations, so the premise is $\beta \equiv (\beta'' \odot z^k)[f \odot (\alpha)^n/z] \equiv \beta'' \odot f^k \odot (\alpha)^{nk}$. Here β'' corresponds to the Γ in the above left rule (the resources of the continuation, besides z^k) and α corresponds to Δ. The full proof of adequacy is in the extended version:

▶ **Theorem 3.2** (Logical adequacy for n-use variables). *$x_1 :^{k_1} A_1, \ldots, x_n :^{k_n} A_n \vdash C$ iff $x_1 : A_1^*, \ldots, x_n : A_n^* \vdash_{x_1^{k_1} \odot \ldots \odot x_n^{k_n}} C^*$, where A^* translates $A \to^n B$ to $\mathsf{U}_{c.c\odot(x^n)}(x : A^* \mid B^*)$*

3.3 Comonads

Following linear-nonlinear logic [5, 6], we decompose the ! exponential of intuitionistic linear logic as the comonad of an adjunction between "linear" and "cartesian" categories. We start with two modes l (linear) and c (cartesian), along with a commutative monoid $(\otimes, 1)$ on l and a cartesian monoid $(\times, \top)$ on c. Next, we add a context descriptor from c to l $(x : \mathsf{c} \vdash \mathsf{f}(x) : \mathsf{l})$ that we think of as including a cartesian context in a linear context. This generates types $\mathsf{F}_{\mathsf{f}(x)}(x : A_\mathsf{c})\ \mathsf{type}_\mathsf{l}$ and $\mathsf{U}_{x.\mathsf{f}(x)}(\cdot \mid A_\mathsf{l})\ \mathsf{type}_\mathsf{c}$ which are adjoint $\mathsf{F}_{\mathsf{f}(x)}(x : -) \dashv \mathsf{U}_{x.\mathsf{f}(x)}(\cdot \mid -)$. The bijection on hom-sets is defined using FL and UR and their invertibility. The comonad of the adjunction $\mathsf{F}_{\mathsf{f}(x)}(x : \mathsf{U}_{c.\mathsf{f}(c)}(\cdot \mid A))$ is the linear logic $!A$.

In LNL [5], $F(A \times B) \cong F(A) \otimes F(B)$ and $F(\top) \cong 1$ (these properties of F are necessary to prove that $!A$ has weakening and contraction with respect to $\otimes$, for example), which we can add to the mode theory by equations $\mathsf{f}(x \times y) \equiv \mathsf{f}(x) \otimes \mathsf{f}(y)$ and $\mathsf{f}(\top) \equiv 1$. By Theorem 2.2, these equations induce type isomorphisms because all of $F, \otimes, \times$ are represented by F-types in our framework. For example, $F(A \times B) \vdash F(A) \otimes F(B)$ is derived as follows:

$$\dfrac{\dfrac{\mathsf{f}(y \times z) \equiv \mathsf{f}(y) \otimes \mathsf{f}(z) \quad \dfrac{}{y : A, z : B \vdash_{\mathsf{f}(y)} \mathsf{F}_{x.\mathsf{f}(x)}(x : A)}\ \mathsf{FR}^* \quad \dfrac{}{y : A, z : B \vdash_{\mathsf{f}(z)} \mathsf{F}_{x.\mathsf{f}(x)}(x : B)}\ \mathsf{FR}^*}{y : A, z : B \vdash_{\mathsf{f}(y \times z)} \mathsf{F}_{z \otimes w}(z : \mathsf{F}_{\mathsf{f}(x)}(x : A), w : \mathsf{F}_{\mathsf{f}(x)}(x : B))}\ \mathsf{FR}}{x : \mathsf{F}_{y \times z}(y : A, z : B) \vdash_{\mathsf{f}(x)} \mathsf{F}_{z \otimes w}(z : \mathsf{F}_{\mathsf{f}(x)}(x : A), w : \mathsf{F}_{\mathsf{f}(x)}(x : B))}\ \mathsf{FL}}{q : \mathsf{F}_{\mathsf{f}(\mathsf{x})}(x : \mathsf{F}_{y \times z}(y : A, z : B)) \vdash_q \mathsf{F}_{z \otimes w}(z : \mathsf{F}_{\mathsf{f}(x)}(x : A), w : \mathsf{F}_{\mathsf{f}(x)}(x : B))}\ \mathsf{FL}$$

Omitting these equations allows us to describe non-monoidal (or lax monoidal, if we add only one direction) left adjoints: in the extended version, we consider S4 $\Box$ [23, 7], and prove adequacy for it.

3.4 Monads

We model a $\Diamond\, A$ modality [7, 23] with rules

$$\frac{\Gamma \vdash A\ \mathsf{true}}{\Gamma \vdash A\ \mathsf{poss}} \qquad \frac{\Gamma \vdash A\ \mathsf{poss}}{\Gamma \vdash \Diamond\, A\ \mathsf{true}} \qquad \frac{A\ \mathsf{true} \vdash C\ \mathsf{poss}}{\Gamma, \Diamond\, A\ \mathsf{true} \vdash C\ \mathsf{poss}}$$

by a mode theory with two modes t and p and context descriptor $x : \mathsf{t} \vdash \mathsf{g}(x) : \mathsf{p}$; we define $\Diamond_{\mathsf{g}}\, A := \mathsf{U}_{c.\mathsf{g}(c)}(\cdot \mid \mathsf{F}_{\mathsf{g}(x)}(x : A))$. This is always a monad, but it does not automatically have a tensorial strength. For example, if we have a monoid $(\otimes, 1)$ on mode t and try to derive strength

$$\dfrac{\dfrac{\mathsf{g}(x \otimes y) \Rightarrow \beta'[\mathsf{g}(y)/z] \quad x : A, y : \Diamond_{\mathsf{g}}\, B, z : \mathsf{F}_{\mathsf{g}}(B) \vdash_{\beta'} \mathsf{F}_{\mathsf{g}}(A \otimes B)}{x : A, y : \Diamond_{\mathsf{g}}\, B \vdash_{\mathsf{g}(x \otimes y)} \mathsf{F}_{\mathsf{g}}(A \otimes B)}\ \mathsf{UL}}{x : A, y : \Diamond_{\mathsf{g}}\, B \vdash_{x \otimes y} \Diamond_{\mathsf{g}}\, (A \otimes B)}\ \mathsf{UR}$$

we are stuck, because there is no way to rewrite $\mathsf{g}(x \otimes y)$ as a term containing $\mathsf{g}(y)$. If $\otimes$ is affine, then we can weaken away x and take $\beta' = z$ – corresponding to the context-clearing in the left rule for $\Diamond\, A$ – but then in the right-hand premise we will only have access to z, not x, so $\Diamond$ correctly represents a non-strong monad in this setting. In the extended version, we prove adequacy for this and extend the mode theory to express strong monads.

▶ **Theorem 3.3** (Logical adequacy for a monad)**.** *We translate all types at mode* t*, representing* $\Diamond\, A$ *as above. Then* $A_1\ \mathsf{true}, \ldots, A_1\ \mathsf{true} \vdash C\ \mathsf{true}$ *iff* $x_1 : A_1^*, \ldots, x_1 : A_n^* \vdash_{x_1 \otimes \ldots \otimes x_n} C^*$*, and* $A_1\ \mathsf{true}, \ldots, A_n\ \mathsf{true} \vdash C\ \mathsf{poss}$ *iff* $x_1 : A_1^*, \ldots, x_1 : A_n^* \vdash_{\mathsf{g}(x_1 \otimes \ldots \otimes x_n)} \mathsf{F}_{\mathsf{g}}(C^*)$*. The three "native" rules above are* FR*,* UR*, and a composite of* UL *followed by* FL*, respectively.*

3.5 Spatial Type Theory

The spatial type theory for cohesion [31] which motivated this work has an adjoint pair $\flat \dashv \sharp$, where $\flat$ is a comonad and $\sharp$ is a monad, with some additional properties. In the one-variable case [15], we analyzed this as arising from an idempotent comonad[2] in the mode theory: we have a mode c with a cartesian monoid $(\times, \top)$ and a context descriptor $x : \mathsf{c} \vdash \mathsf{r}(x) : \mathsf{c}$ such that $\mathsf{r}(\mathsf{r}(x)) \equiv \mathsf{r}(x)$ and there is a directed transformation $\mathsf{r}(x) \Rightarrow x$. Then we define $\flat A := \mathsf{F}_{\mathsf{r}}(A)$ and $\sharp A := \mathsf{U}_{\mathsf{r}}(A)$. These are adjoint, and the transformation gives the counit $\mathsf{F}_{\mathsf{r}}(A) \vdash A$ and the unit $A \vdash \mathsf{U}_{\mathsf{r}}(A)$. Now that we have a multi-assumptioned logic, we can model the fact that $\flat A$ preserves products by the equational axiom $\mathsf{r}(x \times y) \equiv \mathsf{r}(x) \times \mathsf{r}(y)$. Overall, we encode a simply-typed spatial type theory judgement $x_1 : A_1\ \mathsf{crisp}, \ldots; y_1 : B_1\ \mathsf{coh}, \ldots \vdash C\ \mathsf{coh}$ as $x_1 : A_1, \ldots, y_1 : B_1, \ldots \vdash_{\mathsf{r}(x_1) \times \ldots \times y_1 \times \ldots} C$. As a sequent calculus, the rules from [31] are

$$\frac{A \in \Delta \quad \Delta; \Gamma, A \vdash C}{\Delta; \Gamma \vdash C} \qquad \frac{\Delta; \cdot \vdash A}{\Delta; \Gamma \vdash \flat\, A} \qquad \frac{\Delta, A; \Gamma \vdash C}{\Delta; \Gamma, \flat\, A \vdash C} \qquad \frac{\Delta, \Gamma; \cdot \vdash C}{\Delta; \Gamma \vdash \sharp C} \qquad \frac{\sharp\, A \in \Delta \quad \Delta; \Gamma, A \vdash C}{\Delta; \Gamma \vdash C}$$

In order, these correspond to (1) the action of the contraction and $\mathsf{r}(x) \Rightarrow x$ transformations; (2) FR with weakening, using monoidalness of r in one direction; (3) FL; (4) UR, using monoidalness of r in the other direction and idempotence; (5) UL, with contraction. This provides a satisfying explanation for the unusual features of these rules, such as promoting all cohesive variables to crisp in $\sharp$-right, and eliminating a crisp $\sharp$ in $\sharp$-left, and illustrates how our framework can be used in investigating extensions of homotopy type theory.

[2] In [15], the mode theory was actually an idempotent *monad*. The multicategorical generalization prompted changes in the variance of F and U; for example, $\mathsf{F}_\alpha(\Delta)$ must now be covariant in $\psi \vdash \alpha : r$ for the n-ary Δ to match ψ.

4 Equational Theory on Derivations

In this section we give an equational theory describing $\beta\eta$-equality of derivations. We use this equational theory in the categorical semantics below, and to reason about terms in encoded languages (for example, to prove that a pair of entailments is an isomorphism, we show that the maps compose to the identity up to these equations).

First, we need a notation for derivations of the $\alpha \Rightarrow \beta$ judgement in Figure 1. We assume names for constants are given in the signature Σ, and write 1_α for reflexivity, $s_1; s_2$ for transitivity (in diagrammatic order), and $s_1[s_2/x]$ for congruence. We extend the signature Σ to allow axioms for equality of transformations $s_1 \equiv s_2$ (for two derivations of the same judgement $s_1, s_2 :: \psi \vdash \alpha \Rightarrow_p \beta$), and define equality to be the least congruence closed under those axioms and some associativity, unit, and interchange laws, which are the 2-category axioms extended to the multicategorical case (see the extended version for details). As with equality of context descriptors, we think of all definitions as being parametrized by $\equiv$-equivalence-classes of transformations, not raw syntax.

To simplify the axiomatic description of equality, we use a notation for derivations where the admissible transformation, identity, and cut rules are internalized as explicit rules – so the calculus has the flavor of an explicit substitution one. We write proof terms for these plus the 4 U/F rules (the hypothesis rule for atoms is derivable from these) as follows:

$$\frac{}{\Gamma, x : A \vdash_x x : A} \qquad \frac{s :: \alpha \Rightarrow \beta \quad \Gamma \vdash_\beta d : A}{\Gamma \vdash_\alpha s_*(d) : A} \qquad \frac{\Gamma, x : A \vdash_\beta e : B \quad \Gamma \vdash_\alpha d : A}{\Gamma \vdash_{\beta[\alpha/x]} e[d/x] : B}$$

$$\frac{\Gamma, \Gamma', \Delta \vdash_{\beta[\alpha/x]} d : C}{\Gamma, x : \mathsf{F}_\alpha(\Delta), \Gamma' \vdash_\beta (\mathsf{split}\, \Delta \;=\; x \,\mathsf{in}\, d) : C} \qquad \frac{s :: \beta \Rightarrow \alpha[\gamma] \quad \Gamma \vdash_\gamma d_i \vec{/} x_i : \Delta}{\Gamma \vdash_\beta s_*(d_i \vec{/} x_i) : \mathsf{F}_\alpha(\Delta)}$$

$$\frac{x : \mathsf{U}_{x.\alpha}(\Delta \mid A) \in \Gamma \quad s :: \beta \Rightarrow \beta'[\alpha[\gamma]/z] \quad \Gamma \vdash_\gamma d_i \vec{/} x_i : \Delta \quad \Gamma, z : A \vdash_{\beta'} d' : C}{\Gamma \vdash_\beta s_*(\mathsf{let}\, z \;=\; x(d_i \vec{/} x_i) \,\mathsf{in}\, d) : C}$$

$$\frac{\Gamma, \Delta \vdash_{\alpha[\beta/x]} d : A}{\Gamma \vdash_\beta \lambda\Delta.d : \mathsf{U}_{x.\alpha}(\Delta \mid A)}$$

The equational theory of derivations is the least congruence containing the following equations.

$$\begin{array}{rcl} d[x/x] & \equiv & d \\ x[d/x] & \equiv & d \\ d_1[d_2/x] & \equiv & d_1 \text{ if } x \# d_1 \\ (d_1[d_2/x])[d_3/y] & \equiv & (d_1[d_3/y])[d_2[d_3/y]/x] \end{array} \qquad \begin{array}{rcl} 1_*(d) & \equiv & d \\ (s_1; s_2)_*(d) & \equiv & s_{1*}(s_{2*}(d)) \\ (s_2[s_1/x])_*(d_2[d_1/x]) & \equiv & s_{2*}(d_2)[s_{1*}(d_1)/x] \end{array}$$

$$\begin{array}{rcll} (\mathsf{split}\, \Delta \;=\; x_0 \,\mathsf{in}\, d)[s_*(d_i \vec{/} x_i)/x_0] & \equiv & (1_\beta[s/x_0])_*(d[d_i \vec{/} x_i]) & \mathsf{F}\beta \\ (s_*(\mathsf{let}\, z \;=\; x_0(\vec{d_i}/x_i) \,\mathsf{in}\, d'))[\lambda\Delta.d/x_0] & \equiv & (s[1_\alpha/x_0])_*(d'[(d[\vec{d_i}/x_i])/z]) & \mathsf{U}\beta \\ d :: \Gamma, x : \mathsf{F}_\alpha(\Delta), \Gamma' \vdash_\beta C & \equiv & \mathsf{split}\, \Delta \;=\; x \,\mathsf{in}\, d[1_*(\Delta/\Delta)/x] & \mathsf{F}\eta \\ d :: \Gamma \vdash_\beta \mathsf{U}_{x.\alpha}(\Delta \mid A) & \equiv & \lambda\Delta.(1_*(\mathsf{let}\, z \;=\; x(\Delta/\Delta) \,\mathsf{in}\, z))[d/x] & \mathsf{U}\eta \end{array}$$

In the top-left, the first two equations say that identity is a unit for cut. The third says that non-occurrence of a variable is a projection. The fourth is functoriality of cut. In the top-right, the first two rules say that the action of a transformation is functorial, and the third says that it commutes with cut. The typing in the third rule is $d_1 :: \Gamma \vdash_{\alpha'} A$ and $d_2 :: \Gamma, x : A \vdash_{\beta'} C$ and $s_1 :: \alpha \Rightarrow \alpha'$ and $s_2 :: \beta \Rightarrow \beta'$, so both sides are derivations of as derivations of $\Gamma \vdash_{\beta[\alpha/x]} C$. Finally, we have the $\beta\eta$-laws for F and U. The β laws are the principal cut cases from our cut admissibility proof. The η laws witness left-invertibility of F and right-invertibility of U.

5 Categorical Semantics

In this section, we give a category-theoretic structure corresponding to the above syntax. First, we define a cartesian 2-multicategory as a semantic analogue of the syntax in Figure 1.

▶ **Definition 5.1.** A **(strict) cartesian 2-multicategory** consists of

1. A set $\mathcal{M}_0$ of *objects*.
2. For every object B and every finite list of objects $(A_1, \ldots, A_n)$, a category $\mathcal{M}(A_1, \ldots, A_n; B)$. The objects of this category are *1-morphisms* and its morphisms are *2-morphisms*; we write composition of 2-morphisms as $s_1 \cdot s_2$.
3. For each object A, an identity arrow $1_A \in \mathcal{M}(A; A)$.
4. For any object C and lists of objects $(B_1, \ldots, B_m)$ and $(A_{i1}, \ldots, A_{in_i})$ for $1 \leq i \leq m$, a composition functor $(g, (f_1, \ldots, f_m)) \mapsto g \circ (f_1, \ldots, f_m) : \mathcal{M}(B_1, \ldots, B_m; C) \times \prod_{i=1}^{m} \mathcal{M}(A_{i1}, \ldots, A_{in_i}; B_i) \longrightarrow \mathcal{M}(A_{11}, \ldots, A_{mn_m}; C)$. We write the action of this functor on 2-cells as $d \circ_2 (e_1, \ldots, e_m)$.
5. For any function $\sigma : \{1, \ldots, m\} \to \{1, \ldots, n\}$ and objects $A_1, \ldots, A_n, B$, a *renaming* functor $f \mapsto f\sigma^* : \mathcal{M}(A_{\sigma 1}, \ldots, A_{\sigma m}; B) \to \mathcal{M}(A_1, \ldots, A_n; B)$
6. satisfying some equalities (see the extended version)

The next three definitions will be used to describe the $\Gamma \vdash_\alpha A$ judgement.

▶ **Definition 5.2.** A **functor of cartesian 2-multicategories** $\pi : \mathcal{D} \to \mathcal{M}$ consists of a function $\pi_0 : \mathcal{D}_0 \to \mathcal{M}_0$ and functors $\mathcal{D}(A_1, \ldots, A_n; B) \to \mathcal{M}(\pi_0(A_1), \ldots, \pi_0(A_n); \pi_0(B))$ such that the chosen identities, compositions, and renamings are preserved (strictly). Given a functor π, we write $\mathcal{D}_\alpha(A_1, \ldots, A_n; B)$ for the fiber over $\alpha \in \mathcal{M}(\pi A_1, \ldots, \pi A_n; \pi B)$.

▶ **Definition 5.3.** A functor of cartesian 2-multicategories $\pi : \mathcal{D} \to \mathcal{M}$ is a **local discrete fibration** if each induced functor of ordinary categories $\mathcal{D}(A_1, \ldots, A_n; B) \to \mathcal{M}(\pi A_1, \ldots, \pi A_n; \pi B)$ is a discrete fibration. When π is a local discrete fibration, each fiber is a discrete set.

▶ **Definition 5.4.** If $\pi : \mathcal{D} \to \mathcal{M}$ is a local discrete fibration, then a morphism $\xi \in \mathcal{D}(A_1, \ldots, A_n; B)$ is **opcartesian** if all diagrams of the left-hand form are pullbacks of categories, and a morphism $\xi \in \mathcal{D}(\vec{C}, B, \vec{D}; E)$ is **cartesian at** B if all diagrams of the right-hand form are pullbacks of categories:

$$\begin{array}{ccc} \mathcal{D}(\vec{C}, B, \vec{D}; E) & \xrightarrow{(-)\circ_B \xi} & \mathcal{D}(\vec{C}, \vec{A}, \vec{D}; E) \\ \downarrow{\scriptstyle \pi} & & \downarrow{\scriptstyle \pi} \\ \mathcal{M}(\pi\vec{C}, \pi B, \pi\vec{D}; \pi E) & \xrightarrow[(-)\circ_{\pi B} \pi\xi]{} & \mathcal{M}(\pi\vec{C}, \pi\vec{A}, \pi\vec{D}; \pi E) \end{array} \qquad \begin{array}{ccc} \mathcal{D}(\vec{A}; B) & \xrightarrow{\xi \circ_B (-)} & \mathcal{D}(\vec{C}, \vec{A}, \vec{D}; E) \\ \downarrow{\scriptstyle \pi} & & \downarrow{\scriptstyle \pi} \\ \mathcal{M}(\pi\vec{A}; \pi B) & \xrightarrow[\pi\xi \circ_{\pi B} (-)]{} & \mathcal{D}(\pi\vec{C}, \pi\vec{A}, \pi\vec{D}; \pi E) \end{array}$$

Given $\mu : (p_1, \ldots, p_n) \to q$ in $\mathcal{M}$, we say that π **has μ-products** if for any A_i with $\pi A_i = p_i$, there exists a B with $\pi B = q$ and an opcartesian morphism in $\mathcal{D}_\mu(A_1, \ldots, A_n; B)$. Dually, we say π **has μ-homs** if for any i, any B with $\pi B = q$, and any A_j with $\pi A_j = p_j$ for $j \neq i$, there exists an A_i with $\pi A_i = p_i$ and a cartesian morphism in $\mathcal{D}_\mu(A_1, \ldots, A_n; B)$. We say that π is an **opfibration** if it has μ-products for all μ, a **fibration** if it has μ-homs for all μ, and a **bifibration** if it is both an opfibration and a fibration.

The proofs of our soundness and completeness results are described in Appendix A:

▶ **Theorem 5.5** (Mode theory presents a multicategory)**.** *A mode theory Σ presents a cartesian 2-multicategory $\mathcal{M}$, where $\mathcal{M}_0$ is the set of modes, and an object of $\mathcal{M}(p_1, \ldots, p_n; q)$ is a term $x_1 : p_1, \ldots, x_n : p_n \vdash \alpha : q$ and a morphism of $\mathcal{M}(p_1, \ldots, p_n; q)$ is a structural transformation $s :: \psi \vdash \alpha \Rightarrow_q \beta$, both considered modulo $\equiv$.*

▶ **Theorem 5.6** (Completeness/Syntactic Bifibration). *For a fixed mode theory $\mathcal{M}$, the syntax presents a bifibration $\pi : \mathcal{D} \to \mathcal{M}$, where:*

- *Objects of $\mathcal{D}$ are pairs $(p, A\ \mathsf{type}_p)$;*
- *1-morphisms $\Gamma \to B$, i.e., objects of $\mathcal{D}(\Gamma; B)$, are pairs $(\alpha, d :: \Gamma \vdash_\alpha B)$ (up to $\equiv$);*
- *2-morphisms $(\alpha, d) \to (\alpha', d')$ are structural transformations $s :: \alpha \Rightarrow \alpha'$ such that $s_*(d') \equiv d$;*
- *the μ-products are F-types, and the μ-homs are U-types.*

The functor $\pi : \mathcal{D} \to \mathcal{M}$ is given by first projection on objects and 1-morphisms, and sends 2-morphisms to the underlying structural transformations.

▶ **Theorem 5.7** (Soundness/Interpretation in any bifibration). *Fix a bifibration $\pi : \mathcal{D} \to \mathcal{M}$. Then there is a function $[\![-]\!]$ from types $A\ \mathsf{type}_p$ to $[\![A]\!] \in \mathcal{D}_0$ with $\pi([\![A]\!]) = p$ and from $\equiv$-classes of derivations $x : A_1, \ldots, x_n : A_n \vdash_\alpha C$ to morphisms $d \in \mathcal{D}([\![A_1]\!], \ldots, [\![A_n]\!]; [\![C]\!])$, such that $\pi(d) = \alpha$.*

6 Related and Future Work

We have described a sequent calculus that can express a variety of substructural and modal logics through a suitable choice of mode theory. Our framework builds on many approaches to substructural and modal logic in the literature. Logical rules that act at a leaf of a tree-structured context go back to the Lambek calculus [14]. A rich collection of context structures that correspond to type constructors plays a central role in display logic [4]. The λ-calculus for resource separation [3] is similar to mode theories with one mode, where there is at most one 2-cell between a given pair of 1-cells; at the logical level, our calculus is a unification of this with multimodal adjoint logic [26]. Algebras of resources play a central role in semantics of substructural and modal logics [28] and in their encodings in first-order logic [27], and resources on variables are used to track modalities in Agda's implementation [1] and in linear dependent types [17]. LF representations of modal or substructural logics work by restricting the use of cartesian variables [9]. Relative to all of these approaches, we believe that the analysis of the context structures/resources as a *term* in a base type theory, and the fibrational structure of the derivations over them, is a new and useful observation. For example, rather than needing extra-logical conditions on proof rules to ensure cut admissibility, as in display logic, the conditions are encoded in the language of context descriptors and the definition of types from them. Moreover, none of these existing approaches allow for proof-relevant 2-cells/structural rules, and their presence (and the equational theory we give for them) is important for our applications to homotopy type theory.

A point of contrast with substructural logical frameworks [8, 34, 25] is that logics are "embedded" in our calculus (giving a type translation such that provability in the object logic corresponds to provability in ours), rather than "encoding" the structure of derivations. This way, we obtain cut elimination for object languages as a corollary of framework cut elimination.

Bifibrations have also been used recently to model refinement types and logical relations [18, 19, 11]. Superficially, this seems to be a different use of the same semantic structure, because the base and domain categories of the bifibration play a different role. Here, the base mode theory describes different categories of types and functors (type constructors) between between them, and the domain represents types and terms; whereas in refinement types/logical relations, the base category represents types and terms, and the domain represents a further

notion of predicate/specification. It would be interesting to investigate deeper connections and combine the two notions.

One direction for future work is to continue a preliminary investigation of equational adequacy that is discussed in the extended version, investigating whether the logical adequacy proofs are an isomorphism on $\beta\eta$-classes of derivations – or, phrased semantically, that a bifibration over the mode theory is the usual notion of categorical model for a particular logic (e.g. a bifibration over the ordered logic mode theory presents the free monoidal category). It is generally easy to show that object-language equations are true in the framework. We conjecture that the converse is true for the mode theories we have described here, which says that the "extra" types and judgements available in the framework do not add to the equations between terms in the image of encoded sequents. Proving this is challenging because the equational theory of Section 4 does not itself obviously have the subformula property. We have sketched a proof of equational adequacy for a simple case (ordered logic products), assuming a lemma that the equational theory from Section 4 can be characterized by permuting conversions on cut-free derivations. A related avenue for improvement is that our adequacy proofs require reasoning about the 1- and 2-cells in the mode theory, which we have currently done entirely naïvely, but could possibly benefit from higher-dimensional rewriting techniques.

Additionally, we plan to apply our framework to investigate more extensions of homotopy type theory, such as an internal language for parametrized spectra, and an extension of spatial type theory to differential cohesion [29]. We also plan to consider encodings of programming-focused type theories, such as specialized effect calculi.

A final direction for future work is to extend our framework with first-order quantifiers, structured conclusions (as in classical or display logic), and dependent types, which all seem possible but not obvious. Scaling to dependent types will require more worked examples to understand the patterns of substructural and modal type dependency that a framework should capture.

References

1. Andreas Abel. The next 700 modal type assignment systems. In *International Conference on Types for Proofs and Programs (TYPES)*, 2015.
2. Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2016.
3. Robert Atkey. A λ-calculus for resource separation. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004*, volume 3142 of *Lecture Notes in Computer Science*, pages 158–170. Springer, 2004.
4. Nuel D. Belnap Jr. Display logic. *Journal of Philosophical Logic*, 11:375–417, 1982.
5. Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *Computer Science Logic*, volume 933 of *LNCS*. Springer-Verlag, 1995.
6. Nick Benton and Philip Wadler. Linear logic, monads and the lambda calculus. In *IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1996.
7. G. Bierman and V.C.V. de Paiva. On an intuitionistic modal logic. *Studia Logica*, 65:383–416, 2000.
8. Iliano Cervesato and Frank Pfenning. A linear logical framework. *Information and Computation*, 179(1):19–75, 2002.
9. Karl Crary. Higher-order representation of substructural logics. In *ACM SIGPLAN International Conference on Functional Programming*, pages 131–142. ACM, 2010.

10 Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. The structure of exponentials: Uncovering the dynamics of linear logic proofs. In *Kurt Godel Colloquium*, LNCS, pages 159–171. Springer, 1993.

11 Neil Ghani, Patricia Johann, Fredrik Nordvall Forsberg, Federico Orsanigo, and Tim Revell. Bifibrational functorial semantics for parametric polymorphism. In *Proceedings of Mathematical Foundations of Program Semantics*, 2015.

12 Claudio Hermida. Fibrations for abstract multicategories. Fields Institute Communications; available from `http://sqig.math.ist.utl.pt/pub/HermidaC/fib-mul.pdf`, 2002.

13 Fritz Hörmann. Fibered multiderivators and (co)homological descent. Available from `http://arxiv.org/abs/1505.00974`, 2015.

14 Joachim Lambek. The mathematics of sentence structure. *The American Mathematical Monthly*, 65:154–170, 1958.

15 Daniel R. Licata and Michael Shulman. Adjoint logic with a 2-category of modes. In *Logical Foundations of Computer Science*, 2016.

16 Daniel R. Licata, Michael Shulman, and Mitchell Riley. A fibrational framework for substructural and modal logics (extended version). Available from `http://dlicata.web.wesleyan.edu/`, 2017.

17 Conor McBride. I got plenty o' nuttin'. *A List of Successes That Can Change the World: Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, pages 207–233, 2016.

18 Paul-André Melliès and Noam Zeilberger. Functors are type refinement systems. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2015.

19 Paul-André Melliès and Noam Zeilberger. A bifibrational reconstruction of lawvere's presheaf hyperdoctrine. In *IEEE Symposium on Logic in Computer Science*, 2016.

20 Vivek Nigam and Dale Miller. Algorithmic specifications in linear logic with subexponentials. In *Principles and Practice of Declarative Programming*, 2009.

21 Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

22 Frank Pfenning. A structural proof of cut elimination and its representation in a logical framework. Technical Report CMU-CS-94-218, Department of Computer Science, Carnegie Mellon University, 1994.

23 Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11:511–540, 2001.

24 Jason Reed. Names are (mostly) useless: Encoding nominal logic programming techniques with use-counting and dependent types. Talk at Working on Mechanizing Metatheory (WMM), 2008.

25 Jason Reed. *A Hybrid Logical Framework*. PhD thesis, Carnegie Mellon University, 2009.

26 Jason Reed. A judgemental deconstruction of modal logic. Available from `www.cs.cmu.edu/~jcreed/papers/jdml.pdf`, 2009.

27 Jason Reed and Frank Pfenning. A constructive approach to the resource semantics of substructural logics. Available from `http://www.cs.cmu.edu/~jcreed/papers/rp-substruct.pdf`, 2009.

28 Greg Restall. *An introduction to substructural logics*. Routledge, 2000.

29 Urs Schreiber. Differential cohomology in a cohesive infinity-topos. *arXiv*, 2013. `arXiv:1310.7930`.

30 Urs Schreiber and Michael Shulman. Quantum gauge field theory in cohesive homotopy type theory. In *Workshop on Quantum Physics and Logic*, 2012.

31 Michael Shulman. Brouwer's fixed-point theorem in real-cohesive homotopy type theory. arXiv:1509.07584, 2015.

32 Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations Of Mathematics*. Available from `homotopytypetheory.org/book`, 2013.

33 Vladimir Voevodsky. A very short note on homotopy λ-calculus. `http://www.math.ias.edu/vladimir/files/2006_09_Hlambda.pdf`, September 2006.

34 Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.

A Technical Appendix: Categorical Semantics

To illustrate the connection between the syntax and the semantics, we sketch the proofs of soundness and completeness; full details are available in the extended version.

Proof of Theorem 5.6. Our goal is to construct a bifibration $\pi : \mathcal{D} \to \mathcal{M}$ from the syntax.

First, a mode theory Σ presents a cartesian 2-multicategory $\mathcal{M}$, where $\mathcal{M}_0$ is the set of modes, and an object of $\mathcal{M}(p_1, \ldots, p_n; q)$ is a term $x_1 : p_1, \ldots, x_n : p_n \vdash \alpha : q$ and a morphism of $\mathcal{M}(p_1, \ldots, p_n; q)$ is a structural transformation $s :: \psi \vdash \alpha \Rightarrow_q \beta$, both considered modulo $\equiv$.

Next, we construct the domain category $\mathcal{D}$ as indicated in the theorem statement: an object is a pair (p, A) where A is a type of mode p; a 1-cell $(\psi, \Gamma) \to (p, A)$ is a pair (α, d) where $\psi \vdash \alpha : p$ and d is a derivation of $\Gamma \vdash_\alpha A$; and a 2-cell $(\alpha, d) \Rightarrow (\alpha', d')$ is a structural transformation $s :: \alpha \Rightarrow \alpha'$ such that $s_*(d') \equiv d$. We write just A and d and s for objects and 1-cells and 2-cells, leaving the underlying modes and mode morphisms implicit, and we also omit variable names from sequents.

Composition of 1-morphisms is defined by iterating cut: given $g :: (x_1 : B_1, \ldots, x_m : B_m \vdash_\alpha C)$ and $f_i :: (A_{i1}, \ldots, A_{in_i} \vdash_{\beta_i} B_i)$ we set

$$g \circ (f_1, \ldots, f_n) := (\alpha[\beta_1/x_1, \ldots, \beta_m], g[f_1/x_1, \ldots, f_m/x_m])$$

That the latter derivation lies over $\alpha[\beta_1/x_1, \ldots, \beta_m]$ follows from the cut and weakening principles.

For the action of these composition functors on 2-morphisms, suppose we are given 1-morphisms

$$\begin{aligned}
&d :: x_1 : B_1, \ldots, x_m : B_m \vdash_\alpha C \\
&d' :: x_1 : B_1, \ldots, x_m : B_m \vdash_{\alpha'} C \\
&e_i :: A_{i1}, \ldots, A_{in_i} \vdash_{\beta_i} B_i \\
&e'_i :: A_{i1}, \ldots, A_{in_i} \vdash_{\beta'_i} B_i
\end{aligned}$$

and 2-morphisms $S : (\alpha, d) \Rightarrow (\alpha', d')$ and $T_i : (\beta_i, e_i) \to (\beta'_i, e'_i)$ such that S has underlying transformation $s :: \alpha \Rightarrow \alpha'$ and the T_i have underlying transformations $t_i :: \beta_i \Rightarrow \beta'_i$ respectively. This means that $d \equiv s_*(d')$ and $e_i \equiv (t_i)_*(e'_i)$ for all i. The composite $S \circ_2 (T_1, \ldots, T_m)$ is the 2-morphism given by the underlying transformation $s[t_1/x_1, \ldots, t_m/x_m]$. This is a valid 2-morphism $d[e_1/x_1, \ldots, e_m/x_m] \Rightarrow d'[e'_1/x_1, \ldots, e'_m/x_m]$ because

$$\begin{aligned}
&(s[t_1/x_1, \ldots, t_m/x_m])_*(d'[e'_1/x_1, \ldots, e'_m/x_m]) \\
\equiv &(s[t_1/x_1, \ldots, t_{m-1}/x_{m-1}])_*(d'[e'_1/x_1, \ldots, e'_{m-1}m/x_{m-1}])[(t_m)_*(e'_m)/x_m] \\
&\vdots \\
\equiv &s_*(d')[(t_1)_*(e'_1)/x_1, \ldots, (t_m)_*(e'_m)/x_m] \\
\equiv &d[e_1/x_1, \ldots, e_m/x_m]
\end{aligned}$$

as required, where we have repeatedly applied the equation

$$(s_2[s_1/x])_*(d_2[d_1/x]) \equiv s_{2*}(d_2)[s_{1*}(d_1)/x]\,.$$

The unit and associativity laws for 1-morphisms in $\mathcal{D}$ follow from the first set of equations for derivations, and from the definition of multi-variable substitution as iterated cut. For 2-morphisms, they follow as composition of 2-morphisms is simply composition of the underlying transformations in the mode theory.

The cartesian structure in $\mathcal{D}$ is given by the admissible rules for weakening-over-weakening, exchange-over-exchange and contraction-over-contraction, from which all renamings can be made. These rules also all preserve the underlying mode morphisms in the correct way to make π functorial.

The next step is to show that π is a local discrete fibration. Suppose we have a context Γ and object B. We must show that the functor $\pi : \mathcal{D}(\Gamma; B) \to \mathcal{M}(\pi\Gamma; \pi B)$ is a discrete fibration. Let $\alpha, \alpha' \in \mathcal{M}(\pi\Gamma; \pi B)$ be mode morphisms and suppose we have a transformation $s :: \alpha \Rightarrow \alpha'$ between them. Any 2-morphism in $\mathcal{D}(\Gamma; B)$ lying over s must clearly have s as the underlying transformation. Given a lift $d' :: \Gamma \vdash_{\alpha'} B$ of α', then we can consider s as a 2-morphism $(\alpha, s_*(d')) \Rightarrow (\alpha', d')$ over s, whose domain is the action of s on d', $s_*(d')$, as expected. The equational condition $s_*(d) \equiv s_*(d)$ is trivially satisfied, and in fact forces $s_*(d)$ as the only possible choice of domain, so the lift is unique. So π is a local discrete fibration.

We now show that π is an opfibration, i.e., has α-homs for all mode morphisms $\psi \vdash \alpha : q$. Suppose we have lifts for the modes in ψ, i.e., a context Δ with $\pi\Delta = \psi$. We define the opcartesian lift of α to be $\mathsf{FR}^* :: \Delta \vdash_\alpha \mathsf{F}_\alpha(\Delta)$, the generating map from Δ to $\mathsf{F}_\alpha(\Delta)$ that lives over α, which is an instance of the F right rule where both premises are the identity. To verify that this is an opcartesian morphism, we must show that all squares of the form

$$\begin{array}{ccc} \mathcal{D}(\Gamma, \mathsf{F}_\alpha(\Delta), \Gamma'; C) & \xrightarrow{-[\mathsf{FR}^*/x_0]} & \mathcal{D}(\Gamma, \Delta, \Gamma'; C) \\ \downarrow{\scriptstyle \pi} & & \downarrow{\scriptstyle \pi} \\ \mathcal{M}(\pi\Gamma, q, \pi\Gamma'; \pi C) & \xrightarrow[-[\alpha/x_0]]{} & \mathcal{M}(\pi\Gamma, \psi, \pi\Gamma'; \pi C) \end{array}$$

are pullbacks of categories. For this we will use the following characterisation: a diagram of categories

$$\begin{array}{ccc} \mathcal{A} & \xrightarrow{H} & \mathcal{B} \\ {\scriptstyle K}\downarrow & & \downarrow{\scriptstyle F} \\ \mathcal{C} & \xrightarrow[G]{} & \mathcal{D} \end{array}$$

is a pullback diagram iff

- For every pair of objects $b \in \mathcal{B}$ and $c \in \mathcal{C}$ with $Fb = Gc$, there is a unique object $a \in \mathcal{A}$ such that $Ha = b$ and $Ka = c$; and,
- For every pair of morphisms $f \in \mathcal{B}(b, b')$ and $g \in \mathcal{C}(c, c')$ with $Fb = Gc$ and $Fb' = Gc'$ and $Ff = Gg$, there is a unique morphism $\theta \in \mathcal{A}$ such that $H\theta = f$ and $K\theta = g$. The domain and codomain of θ are fixed by the previous property.

First, we show the property for objects. Suppose we have an object $d \in \mathcal{D}(\Gamma, \Delta, \Gamma'; C)$ and $\beta \in \mathcal{M}(\pi\Gamma, q, \pi\Gamma'; \pi C)$ such that $\pi(d) = \beta[\alpha/x_0]$. This simply states that d is of the form $d :: \Gamma, \Delta, \Gamma' \vdash_{\beta[\alpha/x_0]} C$. We must produce a unique object $e \in \mathcal{D}(\Gamma, \mathsf{F}_\alpha(\Delta), \Gamma'; C)$ such

that $\pi(e) = \beta$ and $e[\mathsf{FR}^*/x_0] \equiv d$. We take as our e the derivation $\mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, d$. This lies over β, and we calculate

$$\begin{aligned} e[\mathsf{FR}^*/x_0] &= (\mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, d)[1_{\alpha *}(x/x)/x_0] \\ &\equiv (1_\beta[1_\alpha/x_0])_*(d[\vec{x/x}]) \\ &\equiv (1_{\beta[\alpha/x_0]})_*(d) \\ &\equiv d \end{aligned}$$

by the β-law for F and unit laws. It remains to show uniqueness. Suppose we have some derivation e' such that $\pi(e') = \beta$ and $e'[\mathsf{FR}^*/x_0] \equiv d$. By the η-law for F, we have

$$e' \equiv \mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, e'[\mathsf{FR}^*/x_0] \equiv \mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, d = e$$

as required.

We now turn to the pullback property for morphisms. Let $\beta, \beta' \in \mathcal{M}(\pi\Gamma, q, \pi\Gamma'; \pi C)$ and let $s :: \beta \Rightarrow \beta'$ be a morphism. Further suppose that we have derivations $d :: \Gamma, \Delta, \Gamma' \vdash_{\beta[\alpha/x_0]} C$ and $d' :: \Gamma, \Delta, \Gamma' \vdash_{\beta'[\alpha/x_0]} C$ such that $(s[1_\alpha/x_0])_*(d') \equiv d$. This describes a morphism $T : d \Rightarrow d'$ in $\mathcal{D}(\Gamma, \mathsf{F}_\alpha(\Delta), \Gamma'; C)$ that lies over $s[1_\alpha/x_0]$. This latter transformation is the result of applying the functor $-[\alpha/x_0]$ to s.

We now must find a morphism S in $\mathcal{D}(\Gamma, \Delta, \Gamma'; C)$ that lies over s, and such that the functor $-[\mathsf{FR}^*/x_0]$ applied to the morphism S yields T. We know that for S to lie over s, its underlying structural transformation must be s. The action of $-[\mathsf{FR}^*/x_0]$ on S then takes s to $s[1_\alpha/x_0]$ as expected.

By the previous argument for objects, we know that S must have domain $\mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, d$ and codomain $\mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, d'$. We can verify that choosing the underlying transformation s gives a well-defined 2-morphism $S : (\mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, d) \Rightarrow (\mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, d')$:

$$\begin{aligned} s_*(\mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, d') &\equiv \mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, s_*(\mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, d')[\mathsf{FR}^*/x_0] \\ &\equiv \mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, s_*(\mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, d')[(1_\alpha)_*(\mathsf{FR}^*)/x_0] \\ &\equiv \mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, (s[1_\alpha/x_0])_*(\mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, d'[\mathsf{FR}^*/x_0]) \\ &\equiv \mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, (s[1_\alpha/x_0])_*(d') \\ &\equiv \mathsf{split}\,\Delta \;=\; x_0 \,\mathsf{in}\, d \end{aligned}$$

where we have used the η-law followed by the β-law.

We conclude that all squares of the given form are pullback squares, and so every α has an opcartesian lift. Therefore π is an opfibration. The proof that π is also a fibration is very similar, using U types instead of F types. ◀

Proof of Theorem 5.7. Conversely, we show some cases of the interpretation of the syntax in any bifibration π over the 2-multicategory $\mathcal{M}$ determined by the mode theory.

Since π is a local discrete fibration, the 2-cells of $\mathcal{M}$ act on the fibers. Suppose $\psi \vdash \alpha, \beta : p$ and $s : \alpha \Rightarrow \beta$. We re-use the notation s_* for the induced function (of sets) $\mathcal{D}_\beta(\Gamma; A) \to \mathcal{D}_\alpha(\Gamma; A)$ that sends an object $d \in \mathcal{D}_\alpha(\Gamma; A)$ to the domain of the unique lift of s with codomain d.

The definition of an opfibration of 2-multicategories guarantees that, given a morphism in the mode category $\psi \vdash \alpha : q$ and a set of objects Δ that lies over ψ, there is an opcartesian morphism over α with domain Δ. For each α we choose one such lift and take the codomain of this morphism as our interpretation of $\mathsf{F}_\alpha(\Delta)$. Let us name this opcartesian lift $\zeta_{\alpha,\Delta} : \Delta \to \mathsf{F}_\alpha(\Delta)$. ζ corresponds to the axiomatic FR^*.

We assume a given interpretation of each atomic proposition $[\![P\ \mathsf{type}_p]\!]$ as an object of $\mathcal{D}$ that lies over p. The sequent calculus rules are interpreted as follows (we elide semantic brackets on objects):

- The identity derivation of a sequent $x :: \Gamma \vdash_x A$ is defined to be $[\![x]\!] = 1_A$.
- Given a derivation $d :: \Gamma \vdash_\beta A$ and transformation $s :: \beta' \Rightarrow \beta$, the respect-for-transformations derivation is interpreted as $[\![s_*(d)]\!] = s_*([\![d]\!])$.
- For $d_1 :: \Gamma, x : A, \Gamma' \vdash_\alpha B$ and $d_2 :: \Gamma, \Gamma' \vdash_\beta A$, cut is interpreted as $[\![d_1[d_2/x]]\!] = [\![d_1]\!] \circ_A [\![d_1]\!]$ (writing $e \circ_A f$ for a one-place composition derived from the n-place multicategory composition).
- For FL

$$\frac{\Gamma, \Gamma', \Delta \vdash_{\beta[\alpha/x]} C}{\Gamma, x : \mathsf{F}_\alpha(\Delta), \Gamma' \vdash_\beta M : C}\ \mathsf{FL}$$

  the inductive hypothesis (after an exchange, which preserves the size of the derivations) gives a morphism $[\![d]\!] \in \mathcal{D}_{\beta[\alpha/x]}(\Gamma, \Delta, \Gamma'; C)$ and we must produce a morphism $\mathcal{D}_\beta(\Gamma, \mathsf{F}_\alpha(\Delta), \Gamma'; C)$. By the opcartesian-ness of $\zeta_{\alpha,\Delta}$, the following square is a pullback:

$$\begin{array}{ccc} \mathcal{D}(\Gamma, \mathsf{F}_\alpha(\Delta), \Gamma'; C) & \xrightarrow{(-)\circ\zeta_{\alpha,\Delta}} & \mathcal{D}(\Gamma, \Delta, \Gamma'; C) \\ \downarrow{\scriptstyle \pi} & & \downarrow{\scriptstyle \pi} \\ \mathcal{M}(\pi\Gamma, \pi\mathsf{F}_\alpha(\Delta), \pi\Gamma'; \pi C) & \xrightarrow[(-)\circ\alpha]{} & \mathcal{M}(\pi\Gamma, \pi\Delta, \pi\Gamma'; \pi C) \end{array}$$

  We are given an object of the bottom left (β) and the top right ($[\![d]\!]$), with $\pi[\![d]\!] = \beta \circ_{\pi\mathsf{F}_\alpha(\Delta)} \alpha$. By the above characterization of pullbacks of categories, there is a unique object $[\![d]\!]^{\mathsf{F}}_{\alpha,\Delta} \in \mathcal{D}(\Gamma, \mathsf{F}_\alpha(\Delta), \Gamma'; C)$ so that $\pi([\![d]\!]^{\mathsf{F}}_{\alpha,\Delta}) = \beta$. We take this object to be our interpretation.
- For FR

$$\frac{s : \beta \Rightarrow \alpha[\gamma] \quad \Gamma \vdash_\gamma M : \Delta}{\Gamma \vdash_\beta \mathsf{F}_\alpha(\Delta)}\ \mathsf{FR}$$

  where $\gamma = (\alpha_1, \ldots, \alpha_n)$ and $\Delta = (C_1, \ldots, C_n)$, the first premise is a 2-cell $s : \beta \Rightarrow \alpha \circ (\alpha_1, \ldots, \alpha_n)$, and the second is interpreted as a set of morphisms $[\![d_i]\!] \in \mathcal{D}_{\alpha_i}(\Gamma; C_i)$. We take the interpretation of the conclusion to be $s_*(\zeta_{\alpha,\Delta} \circ ([\![d_1]\!], \ldots, [\![d_n]\!]))$

What remains is to check that the above interpretation function respects the equational theory on derivations (see the extended version). ◀

Arrays and References in Resource Aware ML*

Benjamin Lichtman[1] and Jan Hoffmann[2]

1 **Carnegie Mellon University, Pittsburgh, PA, USA**
`blichtma@alumni.cmu.edu`
2 **Carnegie Mellon University, Pittsburgh, PA, USA**
`jhoffmann@cmu.edu`

Abstract

This article introduces a technique to accurately perform static prediction of resource usage for ML-like functional programs with references and arrays. Previous research successfully integrated the potential method of amortized analysis with a standard type system to automatically derive parametric resource bounds. The analysis is naturally compositional and the resource consumption of functions can be abstracted using potential-annotated types. The soundness theorem of the analysis guarantees that the derived bounds are correct with respect to the resource usage defined by a cost semantics. Type inference can be efficiently automated using off-the-shelf LP solvers, even if the derived bounds are polynomials. However, side effects and aliasing of heap references make it notoriously difficult to derive bounds that depend on mutable structures, such as arrays and references. As a result, existing automatic amortized analysis systems for ML-like programs cannot derive bounds for programs whose resource consumption depends on data in such structures. This article extends the potential method to handle mutable structures with minimal changes to the type rules while preserving the stated advantages of amortized analysis. To do so, we introduce a swap operation for references and arrays that users can use to make programs suitable for automatic analysis. We prove the soundness of the analysis introducing a potential-annotated memory typing, which gathers all unique locations reachable from a reference. Apart from the design of the system, the main contribution is the proof of soundness for the extended analysis system.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Resource Analysis, Functional Programming, Static Analysis, OCaml, Amortized Analysis

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.26

1 Introduction

Several tools currently exist that can automatically derive loop and recursion bounds for imperative programs including COSTA and SACO [1, 3], KoAT [8], CoFloCo [14], SPEED [17], and LOOPUS [35]. These analyses produce impressive results for integer programs, but mutation, cycles, and the absence of type information make it difficult to automatically derive bounds that depend on the sizes of pointer-based data structures. One approach to deal with the problem is to represent the size of data structures with ghost variables [2].

In purely functional programs, reasoning about heap-based data structures is more feasible since there are strong type guarantees on the shape of the data. For example, we know

* This article is based on research that has been supported, in part, by AFRL under DARPA STAC award FA8750-15-C-0082, by NSF under grant 1319671 (VeriQ), and by the Eric and Wendy Schmidt Fund for Strategic Innovation. Any opinions, findings, and conclusions contained in this document are those of the authors and do not necessarily reflect the views of the sponsoring organizations.

that a functional list is immutable and does not have cycles. As a result, there are several techniques that can automatically or semi-automatically derive bounds that depend on the sizes of functional data structures. Automation is often achieved by relying on type systems [11, 12, 29], recurrence relations [6, 13], and automatic amortized resource analysis (AARA) [9, 18, 19, 22, 27]. These techniques work well as long as the programs are purely functional. The only technique that can derive bounds for ML-like programs with references and arrays is the AARA [21] that is implemented in Resource Aware ML (RAML) [19]. While this analysis works well for higher-order programs that use mutable heap structures to store functions, it cannot derive bounds for programs if their execution depends on the size of data structures that are stored in mutable structures.

In this article, we propose an extension of RAML for automatically deriving symbolic resource bounds that depend on the size of data that are stored in references and arrays. We build off of previous work on type-based amortized resource analysis, which has been shown to be able to infer polynomial bounds for functional programs with nested data structures [22, 18, 19]. Like in previous work, we achieve compositionality by integrating the analysis with a standard type system. Type inference is reduced to efficient linear constraint solving and type derivations can be used as certificates that prove the correctness of the bound. Our technique is also parametric in the resource of interest (e.g., analyzing heap usage, clock cycles, and so on) and works for non-monotone resources that can become available during the execution.

The proposed type system is simpler than existing techniques that incorporate AARA in more imperative settings using separation logic [4] or object-oriented types [23]. One advantage is that our technique can be smoothly integrated with the efficient LP-based type inference of RAML. As a result, bound inference is fully automatic. Another advantage is that the design of our resource-annotated types mirror the design decisions of the ML type system, which is the basis of RAML. A disadvantage is that our system is less expressive than other approaches and requires a certain style of programming. However, the advantages that we get from the automation seem to outweigh the disadvantages, and programming with our system is relatively straightforward. For example, as we show in Section 5, we can automatically derive a bound for a graph search algorithm that traverses and mutates a potentially cyclic data structure.

To illustrate the main ideas of our type system, we describe it for a minimal subset of RAML and restrict the technical development to linear bounds. As we argue in Section 6, the proposed techniques carry over to the polynomial and higher-order setting. Apart from the design of the system, our main contribution is the soundness proof of the analysis with respect to an operational cost semantics. To properly calculate the potential of data stored in mutable heap cells at a given program state, we leverage the idea of *memory typing* [36]. This addition is essential to show the soundness of operations like `swap` that involve mutable heap cells.

2 Language Definition and Semantics

We now define a minimal first-order functional language that only contains the features that are relevant to our contributions. The syntax of our language is defined as follows, where $x \in \mathrm{VID}$ (the set of variable identifiers), $f \in \mathrm{FID}$ (the set of function identifiers), $c \in \mathrm{CID}$ (the set of constructor identifiers), and $n \in \mathbb{Z}$.

$$
\begin{aligned}
e ::= {} & x \mid n \mid f(x_1, \ldots, x_n) \mid \text{let } x = e_1 \text{ in } e_2 \mid \text{true} \mid \text{false} \mid () \mid c\,\langle x_1, \ldots, x_k\rangle \\
& \mid \text{match } x \text{ with } c\,\langle x_1, \ldots, x_k\rangle \Rightarrow e_1 \mid e_2 \\
& \mid \text{ref } x \mid !x \mid x_1 := x_2 \mid \text{swap}(x_1, x_2) \mid \text{share } x \text{ as } (x_1, x_2) \text{ in } e \\
& \mid \text{create}(x, e) \mid \text{get}(x_1, x_2) \mid \text{set}(x_1, x_2, x_3) \mid \text{aswap}(x_1, x_2, x_3) \mid \text{length}(x)
\end{aligned}
$$

$$F ::= \cdot \mid (f, \vec{x}, e_f) :: F$$

Function definitions are mutually recursive and given by triples that consist of a function name f, a vector of formal arguments $\vec{x}$, and a function body e_f. Expressions include function calls, constructors $c\,\langle x_1, \ldots, x_k\rangle$, pattern matches, and the usual operations on references and arrays. We use `ref x` to instantiate a reference, `!x` to retrieve the value of a reference, and `x1 := x2` to set the value of a reference. However, to enable data that has been stored in mutable heap cells to be considered in resource analysis, we leverage `swap` operations [34] for references, which combines retrieval and update in one operation. Similarly for arrays, we use `create(n,e)` to initialize an array of length `n` with each cell set to the value expression `e`, `get(A,i)` to retrieve the `i`th value of `A`, `set(A,i,x)` to set the `i`th value of `A` to be `x`, and `aswap(A,i,x)` to combine retrieval and update like with `swap`.

To simplify type rules and proofs, we assume programs are in *share-let normal form*, where term formers are only applied to variables, as much as it does not restrict expressivity. Furthermore, we use an affine type system, so that every bound variable can occur at most once. If some bound value x must be used more than once, the term `share` x `as` (x_1, x_2) `in` e must be used to bind the value of x to x_1 and x_2 in the expression e. We note that in the implementation of RAML, the programmer is not required to either write in share-let normal form or use the `share` expression, as the compiler translates the program into this form before analysis occurs.

Big-Step Operational Cost Semantics

Let *Loc* be an infinite set of memory locations and define the set of values $v \in \mathit{Val}$ to be

$$v ::= \ \ell \mid \text{Null} \mid \texttt{tt} \mid \texttt{ff} \mid (\mathsf{constr}_c, v_1, \ldots, v_k) \mid (\sigma, n)$$

where $\ell \in \mathit{Loc}$ and an array (σ, n) consists of a size $n \in \mathbb{N}$ and a map $\sigma\colon \{0, \ldots, n{-}1\} \to \mathit{Loc}$.

We further define a *heap* H to map locations to values, an *environment* V to map variable identifiers to values, and a *resource metric* $M\colon K \times \mathbb{N} \to \mathbb{Q}$ to describe the resource consumption in each evaluation step of the big-step semantics, where K is a set of constants describing each possible operation in the language. We write M_n^k for $M(k, n)$ and M^k for $M(k, 0)$ in the style of previous work [19]. A metric defines the constant cost of different atomic steps in the cost semantics.

To formalize the notion of tracking resources that can become available during evaluation, we define the *high-water mark* of the resource usage to be the maximal number of resource units under a given metric that are simultaneously used during an evaluation. We use this information in the soundness proof to relate the bound of resource usage generated by the type system to the actual cost of program evaluation.

The operational evaluation rules in Appendix A define the evaluation judgement

$$F, V, H \,{}_M\!\vdash e \Downarrow (v, H') \mid (q, q')$$

where, given a family of functions F, environment $V\colon \text{VID} \rightharpoonup \mathit{Val}$, initial heap $H\colon \mathit{Loc} \rightharpoonup \mathit{Val}$, and resource metric M, the expression e evaluates to the value v and new heap H', requiring $q \in \mathbb{Q}_0^+$ resource units to evaluate and leaving $q' \in \mathbb{Q}_0^+$ resource units available after evaluation. The net resource consumption is $\delta = q - q'$. Note that δ is negative if more resources become available than are consumed during the execution of e.

We define the operation $(q, q') \cdot (p, p')$ to account for an evaluation made up of an evaluation with resource consumption (q, q') followed by an evaluation with resource consumption (p, p') as follows:

$$(q, q') \cdot (p, p') = \begin{cases} (q + p - q',\ p') & \text{if } q' \leq p \\ (q,\ p' + q' - p) & \text{if } q' > p \end{cases}$$

We present selected rules from the operational semantics below. We note that the (E:Fun) and (E:Let) rules are minimally changed from previous work, and the (E:Constr) rule is a generalized form of the rule for specific datatypes in earlier forms of Resource Aware ML.

$$\frac{(f, \vec{y}, e_f) \in F \qquad F, [y_1 \mapsto V(x_1), \cdots, y_n \mapsto V(x_n)], H_M \vdash e_f \Downarrow (v, H') \mid (q, q')}{F, V, H_M \vdash f(x_1, \ldots, x_n) \Downarrow (v, H') \mid M_1^{\text{fun}_n} \cdot (q, q') \cdot M_2^{\text{fun}_n}} \text{(E:Fun)}$$

$$\frac{F, V, H_M \vdash e_1 \Downarrow (v_1, H_1) \mid (q, q') \qquad F, V[x \mapsto v_1], H_1{}_M \vdash e_2 \Downarrow (v_2, H_2) \mid (p, p')}{F, V, H_M \vdash \text{let } x = e_1 \text{ in } e_2 \Downarrow (v_2, H_2) \mid M_1^{\text{let}} \cdot (q, q') \cdot M_2^{\text{let}} \cdot (p, p') \cdot M_3^{\text{let}}} \text{(E:Let)}$$

$$\frac{c_i \in \text{CID} \qquad v = (\mathsf{constr}_{c_i}, V(x_1), \ldots, V(x_k)) \qquad H' = H, \ell \mapsto v \qquad \ell \notin \text{dom}(H)}{F, V, H_M \vdash c_i \langle x_1, \ldots, x_k \rangle \Downarrow (\ell, H') \mid M^{\text{cons}}} \text{(E:Constr)}$$

The rule (E:Fun) can be read as follows. We start evaluating $f(x_1, \ldots, x_n)$ by incurring the constant resource cost $M_1^{\text{fun}_n}$, which depends on the number of function arguments n. If the function f is available in the current context of functions, associated with parameters $y_1, \ldots, y_n$ and body e_f, then e_f is evaluated in an environment where each parameter y_i maps to the value associated with the variable x_i in the calling environment. Moreover, e_f evaluates to the value v and new heap H' with resource cost (q, q'). Then we incur the constant resource cost $M_2^{\text{fun}_n}$ to complete evaluation.

The rule (E:Let) is similar. We first incur the constant resource cost M_1^{let} to begin evaluation, and then in the same environment and heap, evaluate e_1 to the value v_1 and new heap H_1 with resource cost (q, q'). Then, after incurring another constant resource cost M_2^{let}, we evaluate e_2 under the new heap and the same environment with the bound variable x mapping to the value v_1. We thus evaluate e_2 to the value v_2 and heap H_2 with resource cost (p, p'), and then conclude evaluation with the constant cost M_3^{let}.

Lastly, the rule (E:Cons) can be understood as follows. If c_i is a valid constructor identifier, we create a new value $(\mathsf{constr}_{c_i}, V(x_1), \ldots, V(x_n))$ that is a tuple of the constructor name and the values associated with each of its arguments. We then add a fresh memory location ℓ and evaluate to our new value and a heap that has been extended with the new location ℓ pointing to the new value. This all incurs the constant resource cost M^{cons}.

The remaining rules for the operational semantics are presented in Appendix A.

Well-Formed Environments

Our type judgement takes the form $\Sigma; \Gamma \vdash e : T$; under the function context Σ, which describes the types of functions currently in scope, and the variable context Γ, which maps variables to their types, the expression e has type T. The rules for the judgement are presented in

Appendix B. We note that these rules only differ from those discussed in Section 3 in their exclusion of resource annotations, which are developed in the next section.

Given a heap H, value v, and type T, the judgement $H \vDash v : T$ means that the value v under the heap H is well-formed with respect to T. We denote that an environment V and heap H are *well-formed* with respect to a context Γ with $H \vDash V : \Gamma$ if $H \vDash V(x) : \Gamma(x)$ holds for every $x \in \mathrm{dom}(\Gamma)$.

We show that under any evaluation with a well-formed environment, every location in the environment remains well-formed and the return value is also well-formed. We include the proof for Theorem 1 in the technical report [30].

▶ **Theorem 1.** *If* $\Sigma; \Gamma \vdash e : T$*,* $H \vDash V : \Gamma$*, and* $F, V, H_M \vdash e \Downarrow (v, H') \mid (q, q')$ *then* $H' \vDash V : \Gamma$ *and* $H' \vDash v : T$*.*

3 Annotated Types for Linear Resource Analysis

The core idea of AARA is to annotate each program point with a *potential function* which maps sizes of reachable data structures to non-negative numbers. The potential functions must ensure that, for every input and every possible evaluation, the potential at a program point is sufficient to pay for the resource cost of the following transition and the potential at the next point. It then follows that the initial potential function describes an upper bound on the resource consumption of the program.

In the examples in this section, we use a resource metric in which we count the number of constructor applications.

Resource Annotations

To perform AARA for our language, we annotate our types for inductive data structures with non-negative rational numbers $q \in \mathbb{Q}_0^+$, defining the linear resource-annotated data types. To annotate types in a compact manner, we use a function Q for each inductive type that maps constructor names to their associated potential annotations. The data types of our language are defined as follows.

$$A ::= 1 \mid B \mid X \mid \mu X^Q.\{c_i : (A_1, \ldots, A_{k_i})\}_i \mid A \text{ ref} \mid A \text{ array}^q \,.$$

Let $\mathscr{A}_{\text{lin}}$ be the set of linear resource-annotated data types and $\mathscr{T}$ be the set of unannotated types. We use 1 as the unit type, B for booleans, X for type variables, A ref for references with data of type A, and A array for arrays with data of type A. We also use recursive datatypes of the form $\mu X^Q.\{c_i : (A_1, \ldots, A_{k_i})\}_i$, where the type variable X is bound in the types A_1 through A_{k_i} for each constructor c_i.

We define the type $N^q = \mu X^{Q_N}.\{\text{zero} : 1 \mid \text{succ} : X\}$ to be used with arrays, where $Q_N[\text{zero}] = 0$ and $Q_N[\text{succ}] = q$. Additionally, for the purpose of the examples in this section, we use the type $L^q(A) = \mu X^{Q_L}.\{\text{nil} : 1 \mid \text{cons} : (A, X)\}$, where $Q_L[\text{nil}] = 0$ and $Q_L[\text{cons}] = q$. This represents a list containing elements of type A, where each element carries q potential. We also have linear resource-annotated first-order types, which are defined by the following grammar.

$$R ::= (A_1, \ldots, A_n) \xrightarrow{q/q'} A \,.$$

In this notation, $q, q' \in \mathbb{Q}_0^+$ and $A, A_i \in \mathscr{A}_{\text{lin}}$, meaning that q is the constant potential before a call to the function and q' is the constant potential after the call to the function. Let $\mathscr{R}_{\text{lin}}$ denote the set of linear resource-annotated first-order types.

The potential annotation of a type directly determines the potential represented by values of that type. For example, when we consider a list of length n and of type $L^q(T)$, the potential carried by that list is qn, where each element carries q resource units. With the resource metric considered here, we are able to pay for up to q `cons` operations for each element in the list (since each element carries q resource units).

We can more formally define the potential of a value. Let $A \in \mathscr{A}_{\text{lin}}$, let H be a heap, and let $v \in \mathit{Val}$ be a value with $H \vDash v : A$. Then, if $H(\ell) = (\mathsf{constr}_{c_i}, v_1, \ldots, v_{k_i})$ and $A = \mu X^Q.\{\cdots \mid c_i : (A_1, \ldots, A_{k_i}) \mid \ldots\}$, we have that

$$\Phi_H(\ell : A) = Q[c_i] + \sum_{i=1}^{k_i} \Phi_H(v_i : [A/X]A_i)$$

where $[A/X]A_i$ represents the substitution of the type A for every instance of the type variable X in A_i. If $H(\ell) = (\sigma, n)$ and $A = A'$ array^q, then $\Phi_H(\ell : A) = qn$. Otherwise, $\Phi_H(v : A) = 0$.

We establish the following definition for the amount of potential provided by a typing context under a particular environment V and heap H.

$$\Phi_{V,H}(\Gamma) = \sum_{x \in \text{dom}(\Gamma)} \Phi_H(V(x) : \Gamma(x))\,.$$

In particular, we sum over the potential contributed by each variable in scope by the definition presented earlier. However, note that this causes data stored within references and arrays to not be included by the above definition of Φ_H.

In order to relate resource-annotated types with different resource annotations, we define the function $|\cdot| \colon \mathscr{A}_{\text{lin}} \to \mathscr{T}$ that simply removes the potential annotations from a given type. If we have annotated types A and A' such that $|A| = |A'|$, we say that they have the same *underlying types*.

The sharing relation $\curlyvee$ defines how the potential of a variable can be shared by multiple occurrences of that variable. We have $A \curlyvee (A_1, A_2)$ if and only if $|A| = |A_1| = |A_2|$ and for every heap H and value v such that $H \vDash v : A$, $\Phi_H(v : A) = \Phi_H(v : A_1) + \Phi_H(v : A_2)$ holds. The sharing relation $\curlyvee$ is the smallest relation such that the following hold.

$$\frac{A \in \{1, B, N, A' \text{ ref}\}}{A \curlyvee (A, A)}\,(S_1) \qquad \frac{p = q + r}{A \text{ array}^p \curlyvee (A \text{ array}^q, A \text{ array}^r)}\,(S_2)$$

$$\frac{A_i \curlyvee (A_i', A_i'') \qquad \forall 1 \le i \le k.\, P[c_i] = Q[c_i] + R[c_i]}{\mu X^P.\{c_i : (A_1, \ldots, A_k)\}_i \curlyvee (\mu X^Q.\{c_i : (A_1', \ldots, A_k')\}_i, \mu X^R.\{c_i : (A_1'', \ldots, A_k'')\}_i)}\,(S_3)$$

▶ **Example 2.** Consider the standard `append` function for our list type. With our resource metric, the cost of append is n where n is the length of the first argument. To derive a bound in our type system, we assign the function `append` the type $(L^1(T), L^0(T)) \to L^0(T)$. This function can also be assigned the type $(L^2(T), L^1(T)) \to L^1(T)$ and so forth; as long as the annotations represent that one resource unit has been consumed for every element of the first input list and there is enough potential remaining to pay for the potential of the result, the potential annotations are valid. The second annotation can be used for the inner call of append in an expression like `append(append(x,y),z)`.

We now consider the following function that calls `append`.

```
f l = share l as (l1,l2) in let _ = append(l1,[]) in append(l2,[])
```

In this example program, we see the results of reusing a value that carries potential. Since we call `append` on the same list twice, it must have sufficient potential to be iterated over twice. In order to do so, when we share the list between two variables, we split the potential over the two, and thus assign the type $L^2(T) \to L^0(T)$ to `f`. In the type derivation, the variable `l` has type $L^2(T)$, and `l1` and `l2` have type $L^1(T)$.

A *resource-annotated typing judgement* has the form $\Sigma; \Gamma \vdash^{q}_{q'} e : A$. This means that, under environment V and heap H such that $H \vDash V : \Gamma$ holds, as well as a resource-annotated signature Σ and resource-annotated context Γ, the expression e has the resource-annotated data type A. If there are at least $q + \Phi_{V,H}(\Gamma)$ resource units available, then e may be evaluated. Furthermore, if e evaluates to a value v, resulting in an updated heap H', there are more than $q' + \Phi_{H'}(v : A)$ resource units left.

We define a well-typed program to consist of a resource-annotated signature Σ and a family $F = (f, \vec{x}, e_f)_{f \in \mathrm{dom}(\Sigma)}$ of function identifiers $f \in \mathrm{FID}$ with variable identifiers $\vec{x} = x_1, \ldots, x_n \in \mathrm{VID}$ and expression e_f such that $\Sigma; x_1{:}A_1, \ldots, x_n{:}A_n \vdash^{q}_{q'} e_f : A$ for each $(A_1, \ldots, A_n) \xrightarrow{q/q'} A \in \Sigma(f)$.

We present selected type rules related to the example above. The rules (L:Fun) and (L:Share) are unchanged from previous work, and the (L:Constr) and (L:Mat) rules are expanded to handle the more general form of recursive datatypes.

$$\frac{c_i \in \mathrm{CID} \qquad A = \mu X^P.\{\cdots \mid c_i{:}(A'_1, \ldots, A'_k) \mid \cdots\} \qquad \forall 1 \le j \le k.\, A_j = A'_j \vee (A_j = A \wedge A'_j = X)}{\Sigma; x_1{:}A_1, \ldots, x_k{:}A_k \vdash^{q+P[c_i]+M^{\mathrm{cons}}}_{q} c_i\ (x_1, \ldots, x_k) : A} \text{(L:Constr)}$$

$$\frac{\Sigma; \Gamma, x_1{:}A_1, x_2{:}A_2 \vdash^{q}_{q'} e : A \qquad A' \curlyvee (A_1, A_2)}{\Sigma; \Gamma, x{:}A' \vdash^{q}_{q'} \text{share } x \text{ as } (x_1, x_2) \text{ in } e : A} \text{(L:Share)}$$

$$\frac{(A_1, \ldots, A_n) \xrightarrow{q/q'} A \in \Sigma(f)}{\Sigma; x_1{:}A_1, \ldots, x_n{:}A_n \vdash^{q+M_1^{\mathrm{fun}_n}}_{q'-M_2^{\mathrm{fun}_n}} f(x_1, \ldots, x_n) : A} \text{(L:Fun)}$$

The rule (L:Constr) can be understood as follows. Since the construction of a new list element costs M^{cons} resource units, we have to pay M^{cons}, as well as the potential that is available after the evaluation. The potential of each of the components of the constructor is paid for by the context, and the missing potential $P[c_i]$ of the new list element, the constant cost M^{cons}, and the resulting potential q are paid by the intial constant potential $q + P[c_i] + M^{\mathrm{cons}}$.

The rule (L:Share) incurs no resource consumption. However, the type A' of the variable x is split into two new types A_1 and A_2 for the new variables x_1 and x_2, which share the potential associated with x.

Lastly, in the rule (L:Fun), the evaluation of $f(x_1, \ldots, x_n)$ incurs the constant cost $M_1^{\mathrm{fun}_n}$ before the evaluation of the body and $M_2^{\mathrm{fun}_n}$ after the evaluation of the body. Since $(A_1, \ldots, A_n) \xrightarrow{q/q'} A \in \Sigma(f)$, we know that the body evaluates with initial constant potential q. Thus in order to pay for the cost before evaluation, we need to have initial potential $q + M_1^{\mathrm{fun}_n}$ for the whole expression. Similarly, we have remaining constant potential q' after evaluation of the body, so after evaluating the whole expression, we have remaining constant potential $q' - M_2^{\mathrm{fun}_n}$.

Potential in References

We now introduce references by way of the following example, where the function `g'` might use the data in the supplied reference in some unknown way.

```
g l = let r = ref l in
      share r as (r1,r2) in
      let _ = g' r1 in
      append(!r2,[])
```

If we assign a type of the form $L^q(T)\ ref \to 1$ to `g'`, we have a weak contract between `g` and `g'` concerning the usage of the data referenced by `r`. Looking only at the type of `g'`, we cannot know how `g'` consumes the potential of the data referenced by `r`. Therefore, we also cannot know how much potential we have left over on the data in `r` after `g'` executes.

The most straightforward way to address the ambiguity of mutable data usage is to insist that the potential annotations of mutable data must remain invariant. By requiring this, we are guaranteed that we can freely share, pass around, and alias references without being concerned with tracking every possible use of the underlying data. This restriction of invariant potential annotations is precisely in line with the goals of the ML type system. As opposed to carrying around extra information about effectful computation statically, the only information that ML tracks is the unchanging type of data that is stored in the reference. Furthermore, as the number of mutable cells generated in a program scales, so too does the opportunity for aliased references to be passed to some subroutine. If the potential annotations of these references could vary, then the system would have to track all possible combinations of distinct and aliased references in these situations in order to successfully analyze resource usage. As this approach clearly cannot scale, we resort to demanding invariant potential annotations for references.

However, when considering the above example, the action of sharing references that contain lists cannot operate in the same way it does when we directly share lists. We note that when sharing the reference `r`, we cannot split the potential of the underlying data (as we do when sharing a list directly) due to our new requirement of invariant potential annotations. In other words, since `r`, `r1`, and `r2` all point to the same heap location, they all must contain the same potential annotation for the data stored there.

If we suppose now that `g'` consumes the potential stored in `r` in the function `g` from before, we find that we have violated the soundness of our system. In this scenario, we see that the same potential is being used to pay for the call to `append` in `g` as well as for whatever operation occurs in `g'`. As discussed above, in a situation not involving references, this duplication would be explicitly handled by using `share`, but due to our invariant potential annotations for references, it does not in this case. Therefore, our program uses double the amount of potential that the type annotations provide.

Introducing Swap

In order to prevent uncontrolled duplication of potential, we use the `swap` operation to better track usage of data in mutable structures, as used in previous treatments of substructural type systems [34, 32], where it is operationally defined as follows:

`swap (r,l)` $\equiv$ `let x = !r in let _ = (r := l) in x.`

With `swap`, we ensure that in order to use data that has been stored in a reference in a way that consumes potential, it must be "swapped" out for data of the same type that satisfies

the same potential annotation. As a result, for data in a single reference to be used twice, it must be swapped out, explicitly shared, and swapped back in.

▶ **Example 3.** If we use `swap` to access data in mutable structures, then we must take care in situations where two references may be, but are not necessarily, aliased so that we do not mistakenly expect to use data that has already been destructively modified. Consider the following program representing this situation, in which we append two lists after retrieving them from references that are possibly aliased to each other. With this, we demonstrate a programming style that ensures that Resource Aware ML can successfully compute a bound.

```
f (r1, r2) =
  let l2_1 = !r2 in
  let l1 = swap(r1,[]) in
  let l2_2 = swap(r2,[]) in
  match (l2_1,l2_2) with
  | ([],_) -> l1
  | (_,[]) -> share l1 as (l1_1,l1_2) in
              append(l1_1,l1_2)
  | (_,_) ->  append(l1,l2_2)
```

Here, we only perform the normal `append` operation if the data stored in `r2` is a non-empty list before *and* after `l1` is retrieved. Otherwise, if it is non-empty before and empty after, we know that the two references must be aliased, and therefore, to achieve the proper result, we duplicate the data retrieved from the first (and therefore explicitly share its potential) and then append the resulting lists. Therefore, if we assign the type $L^p(T)$ *ref* to `r1` and $L^q(T)$ *ref* to `r2`, we thus assign the type $L^p(T)$ to `l1`, $L^0(T)$ to `l2_1`, and $L^q(T)$ to `l2_2`.

It is allowed in our system to dereference data using the usual `!` operator. However, we ensure in our type system that data retrieved in this way has no potential. As is shown by the above example, while the `swap` operation places a new burden on the programmer, it is not unreasonable to work around the unintended effects it may cause.

An interesting observation is that the potential annotations in function types contain information about aliasing. One possible typing for the function `f` is $(L^1(T)\ \mathit{ref}, L^1(T)\ \mathit{ref}) \to L^0(T)$. Here, `r1` and `r2` are potentially aliased. Another possible typing for the function `f` is $(L^1(T)\ \mathit{ref}, L^0(T)\ \mathit{ref}) \to L^0(T)$. This typing implies that the arguments are not aliased; if they were, they would break the invariant of our type system discussed earlier. We now present the type rules that are related to references.

$$\frac{}{\Sigma; x{:}A \left|^{q+M^{\text{ref}}}_{q}\right. \text{ref}\ x : A\ \text{ref}}\ \text{(L:Ref)} \qquad \frac{}{\Sigma; \Gamma, x_1{:}A\ \text{ref}, x_2{:}A \left|^{q+M^{\text{assign}}}_{q}\right. x_1 := x_2 : 1}\ \text{(L:Assign)}$$

$$\frac{|A'| = |A| \qquad A \curlyvee (A, A)}{\Sigma; x{:}A'\ \text{ref} \left|^{q+M^{\text{dref}}}_{q}\right. !x : A}\ \text{(L:DRef)} \qquad \frac{}{\Sigma; x_1{:}A\ \text{ref}, x_2{:}A \left|^{q+M^{\text{swap}}}_{q}\right. \text{swap}(x_1, x_2) : A}\ \text{(L:Swap)}$$

The rule (L:Ref) can be read as follows. Our context contains just the variable $x{:}A$. We thus create a reference of type A ref. Since the operational semantics state that the cost of evaluating such a reference is M^{ref}, we need initial constant potential $q + M^{\text{ref}}$ to pay for this cost and the remaining constant potential q. The rules (L:Assign) and (L:Swap) are similar.

(L:DRef) is also similar, but additionally restricts the type of the dereferenced value. The premise $A \curlyvee (A, A)$ requires that the type have a potential annotation of zero, thus ensuring that the return value has no potential. The first premise simply ensures that the type of the reference and the type of the dereferenced value only differ in their potential annotations.

Potential for Arrays

The concerns and techniques presented above extend cleanly from references to arrays. As before, for each cell in an array, we cannot always tell statically how much potential has been used up by effectful subroutines. Therefore, we maintain the same requirement of invariant potential for every cell in the array, and use the operation `aswap(A,i,x)`, which is defined as follows:

```
aswap(A,i,x) ≡ share A as (A1,A2) in share i as (i1,i2) in
               let x' = get(A1,i1) in let _ = set(A2,i2,x) in x'
```

The relevant rules follow.

$$\frac{\Sigma; \emptyset \left|^{p-r}_{0}\right. e : A}{\Sigma; x{:}N^{p} \left|^{q+M^{\text{create}}}_{q}\right. \text{create}(x, e) : A \text{ array}^{r}} \text{(L:Create)}$$

$$\frac{|A'| = |A| \qquad A \curlyvee (A, A)}{\Sigma; x_1{:}A' \text{ array}^{p}, x_2{:}N^{r} \left|^{q+M^{\text{get}}}_{q}\right. \text{get}(x_1, x_2) : A} \text{(L:Get)}$$

$$\frac{}{\Sigma; x_1{:}A \text{ array}^{p}, x_2{:}N^{r}, x_3{:}A \left|^{q+M^{\text{set}}}_{q}\right. \text{set}(x_1, x_2, x_3) : 1} \text{(L:Set)}$$

$$\frac{}{\Sigma; x_1{:}A \text{ array}^{p}, x_2{:}N^{r}, x_3{:}A \left|^{q+M^{\text{aswap}}}_{q}\right. \text{aswap}(x_1, x_2, x_3) : A} \text{(L:Aswap)}$$

The rule (L:Create) requires the user to specify a default expression to initialize each cell of the new array. This expression is then evaluated using the potential from the number specifying the length of the array. Since this pays for the evaluation of each cell, the system simply requires enough initial constant potential to pay for the constant resource cost, M^{create}, as well as the remaining constant potential, q. Otherwise, (L:Get), (L:Set), and (L:Aswap) are analogous to their counterparts for references. The remaining typing rules are provided in Appendix C.

4 Soundness

We now show that the operational semantics and linear resource-annotated typing rules cohere, proving that type derivations establish correct bounds. We claim that if an expression e evaluates to a value v in a well-formed environment then the initial potential of the context is an upper bound on the watermark of the resource usage. Furthermore, the difference between the initial and final potential is an upper bound on the consumed resources.

To ensure that aliased references are not double-counted when determining the potential that they contribute, we use a *memory typing* Δ: $Loc \to \mathscr{A}_{\text{lin}}$. Under a given context Γ, environment V, and heap H, Δ maps all locations in the heap pointed to by references contained in Γ to the types of their values. This is similar to the usage of store typing in previous work on mutable structures in substructural settings [36]. We represent this relationship with the judgement $H, \Delta \vDash V : \Gamma$ and the operator $\otimes$ defined by the rules in Figure 1, using $\emptyset$ to represent the empty reference collection.

The most interesting rules in Figure 1 are (Δ:Ref), (Δ:Array), (Δ:Constr), and (Δ:Full). The first of these can be read as follows. When checking that the location ℓ is well-formed at

$$\frac{}{\emptyset \otimes \Delta = \Delta}\ (\otimes : \text{U1}) \qquad \frac{}{\Delta \otimes \emptyset = \Delta}\ (\otimes : \text{U2})$$

$$\frac{\forall i \in \{1,2\}.\forall \ell \in \text{dom}(\Delta_i).\Delta(\ell) = \Delta_i(\ell) \qquad \forall \ell \in \text{dom}(\Delta).(\ell \in \text{dom}(\Delta_1)) \vee (\ell \in \text{dom}(\Delta_2))}{\Delta_1 \otimes \Delta_2 = \Delta}\ (\otimes : \text{C})$$

$$\frac{}{H, \Delta \vDash \texttt{tt} : B}\ (\Delta : \text{True}) \qquad \frac{}{H, \Delta \vDash \texttt{ff} : B}\ (\Delta : \text{False}) \qquad \frac{}{H, \Delta \vDash \text{Null} : 1}\ (\Delta : \text{Unit})$$

$$\frac{\begin{array}{c} H(\ell) = (\mathsf{constr}_{c_i}, v_1, \ldots, v_k) \qquad H' = H \setminus \ell \\ A = \mu X^Q.\{\cdots \mid c_i{:}(A_1, \ldots, A_k) \mid \cdots\} \\ H', \Delta_1 \vDash v_1 : [A/X]A_1 \quad \ldots \quad H', \Delta_k \vDash v_k : [A/X]A_k \qquad \Delta = \bigotimes_{1 \le j \le k} \Delta_i \end{array}}{H, \Delta \vDash \ell : \mu X^Q.\{\cdots \mid c_i{:}(A_1, \ldots, A_k) \mid \cdots\}}\ (\Delta : \text{Constr})$$

$$\frac{H(\ell) = \ell' \qquad H, \Delta' \vDash \ell' : A \qquad \Delta = \Delta' \otimes \{\ell' \mapsto A\}}{H, \Delta \vDash \ell : A \text{ ref}}\ (\Delta : \text{Ref})$$

$$\frac{H(\ell) = (\sigma, n) \qquad \forall i < n.H, \Delta_i' \vDash H(\sigma(i)) : A \qquad \Delta_i = \Delta_i' \otimes \{\sigma(i) \mapsto A\} \qquad \Delta = \bigotimes_{0 \le i < n} \Delta_i}{H, \Delta \vDash \ell : A \text{ array}^q}\ (\Delta{:}\text{Array})$$

$$\frac{\text{dom}(\Gamma) \subseteq \text{dom}(V) \qquad \forall x \in \text{dom}(\Gamma).H, \Delta_x \vDash V(x) : \Gamma(x) \qquad \Delta = \bigotimes_{x \in \text{dom}(\Gamma)} \Delta_x}{H, \Delta \vDash V : \Gamma}\ (\Delta{:}\text{Full})$$

Figure 1 Rules Defining the $\otimes$ operator and the well-formed reference collection.

type A ref under heap H, we first ensure that the location points to some other location ℓ' and then check that ℓ' is well-formed at type A under the same heap. If this check gives us back the memory typing Δ', we add the mapping $\ell' \mapsto A$ to Δ' to make Δ. As discussed above, we do this because there exists a reference that points to ℓ', and so we track it as intended. We thus return Δ as the memory typing for the location ℓ. The (Δ:Array) rule is similar, but we check each location pointed to by the array and combine the resulting memory typings. Likewise, the (Δ:Constr) rule checks that each value included in the constructor is well-formed and combines the the resulting memory typings. Lastly, the (Δ:Full) rule checks each value included in the environment V is well-formed and returns the combination of the constructed memory typings.

Note that $H, \Delta \vDash V : \Gamma$ implies $H \vDash V : \Gamma$. Furthermore, we intentionally allow extra locations to be present in Δ, as this simplifies the soundness proof without contributing extra potential. We now define the potential of Δ as follows:

$$\Phi_H^D(\Delta) = \sum_{\ell \in \text{dom}(\Delta)} \Phi_H(\ell : \Delta(\ell))\,.$$

By defining the potential of mutable cells in this way, we ensure that every reachable heap cell is counted exactly once, thereby disallowing the possibility of aliasing leading to extra potential.

Given this additional method of contributing potential, the typing judgement $\Sigma; \Gamma \vdash^{q}_{q'} e : A$ now means that there must be $q + \Phi_{V,H}(\Gamma) + \Phi_H^D(\Delta)$ resource units available to evaluate e, and after evaluation to a value v, there will be $q' + \Phi_{H'}(v : A) + \Phi_{H'}^D(\Delta_{\text{ret}})$ units available, where $H, \Delta \vDash V : \Gamma$ and $H', \Delta_{\text{ret}} \vDash v : A$. We can now formally state our soundness claim.

▶ **Theorem 4.** *Let $H, \Delta \vDash V : \Gamma$ and $\Sigma; \Gamma \vdash^{q}_{q'} e : A$ hold. If $F, V, H \vdash_M e \Downarrow (v, H') \mid (p, p')$ then there exists some Δ_{ret} such that the following hold:*

- $p \leq \Phi_{V,H}(\Gamma) + \Phi^D_H(\Delta) + q$
- $p - p' \leq \Phi_{V,H}(\Gamma) + \Phi^D_H(\Delta) + q - (\Phi_{H'}(v{:}A) + \Phi^D_{H'}(\Delta \otimes \Delta_{ret}) + q')$
- $H', \Delta \vDash V : \Gamma$ *and* $H', \Delta_{ret} \vDash v : A$

Theorem 4 is proved by a nested induction on the derivation of the evaluation judgement and the type judgement, with the latter being needed due to the structural rules. We present the proof of Theorem 4 in the technical report [30]. Moreover, we note that the proof is similar to the soundness proof for resource analysis for the language without mutable heap cells [18].

The soundness proof uses Lemma 5 to show the soundness of the rule L:Let, stating that the potential of a context is invariant during the evaluation.

▶ **Lemma 5.** *Let* $H, \Delta \vDash V : \Gamma$, $\Sigma; \Gamma \vdash^{q}_{q'} e : A$, *and* $F, V, H \vdash_M e \Downarrow (v, H') \mid (p, p')$. *It follows that* $\Phi_{V,H}(\Gamma) = \Phi_{V,H'}(\Gamma)$.

Proof. We get that $H' \vDash V : \Gamma$ by Theorem 1. Thus, the lemma follows directly by this fact and the definition of the potential Φ. ◀

5 Graphs and Graph Search

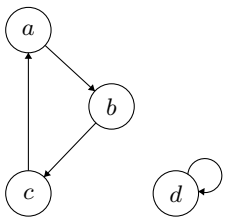
We now discuss graph search as an example of reference usage in RAML. Rather than explicitly showing occurrences of sharing, we use standard OCaml syntax.

We use the following set of user defined types as our graph representation.

```
type 'a node = Not_Visited of 'a * 'a node ref list
             | Visited of 'a * 'a node ref list
             | TEMP

type 'a graph = 'a node ref list
```

Here, we represent a graph as an adjacency list, where each node contains some data, as well as a list of references to the nodes to which it has edges. We wrap each node by either `Not_Visited` or `Visited` so that we can consume the potential of the list of children and then place the node back in its reference. We now build the following graph using the function `make_graph`.




```
let make_graph () : int graph =
  let c_ref = ref (Node (Not_Visited (3, []))) in
  let b_ref = ref (Node (Not_Visited(2, [c_ref]))) in
  let a_ref = ref (Node (Not_Visited(1, [b_ref]))) in
  let _ = swap(c_ref, Node (Not_Visited(3, [a_ref]))) in
  let d_ref = ref (Node (Not_Visited (4, []))) in
  let _ = swap(d_ref, Node (Not_Visited (4, [d_ref]))) in
  [a_ref; b_ref; c_ref; d_ref]
```

We now show how to write DFS over this graph representation. In this implementation, we use the function `iter : ('a -> 1) -> 'a list -> ()` which sequentially applies a function to every element of a list. This implementation could easily be extended to apply a function to the data at each node, but here we simply traverse the graph.

At a not-yet-visited node in the graph, this function will iterate over its list of out-edges. For each, it will swap the node out of its particular reference and replace it with `TEMP`, recursively traverse that node and its respective out-edges, and then place the `Visited` version of that node back into the reference when it has completed its search on that subtree.

As a result, if the search finds `TEMP` within a reference, it must have been swapped out further up the current call stack and therefore must have already been visited.

```
let rec DFS(n : int node) : int node =
  match n with
  | TEMP -> n (* node was visited further up current call stack *)
  | Visited _ -> n (* node was visited in diff. branch of search *)
  | Not_Visited (d,cs) ->
    let _ = iter (fun n_ref ->
              let n' = swap(n_ref, TEMP) in
              let n'' = DFS n' in
              n_ref := n'') cs in
    Visited (d,cs)
```

We now consider tracking the number of recursive calls as a resource metric. Due to the structure of the `node` datatype, the list in a not-visited node is able to carry a different amount of potential than the list in a visited node. Therefore, we can define the type *node* as follows in order to pay for one full run of *DFS*.

$$A\ node = \mu X^{[0,0,0]}.\{\text{Not\_Visited} : (A, L^1(X\ ref)) \mid \text{Visited} : (A, L^0(X\ ref)) \mid \text{TEMP} : 1\}\,.$$

Since the list contained within the `Not_Visited` constructor has a potential annotation of 1, each element can pay for one recursive call to DFS. When looking at the code, we see that this is exactly what occurs.

After running our DFS operation on every member of the list of nodes produced by `make_graph`, we have flipped each node to be marked with `Visited` and therefore have exhausted all the potential in each of the nodes. If we wish to refresh the graph and set each node back to `Not_Visited` with the intent to run our graph search again, we could once again iterate over the top-level list of nodes as returned by `make_graph`. However, if our program did this, then the potential annotation of the return type of `make_graph` would have to be increased, where our metric now tracks the number of times a node is changed from `Visited` to `Not_Visited` and vice versa. In summary, the type of `make_graph` is as follows (assuming we have some type *int*).

$$make\_graph\ ((): 1) : L^1(int\ node) \qquad \text{[With no refresh]}$$
$$make\_graph\ ((): 1) : L^3(int\ node) \qquad \text{[With one refresh]}$$

6 Type Inference for Linear and Polynomial Bounds and Higher-Order Functions

The main advantage of the resource-annotated type system we introduce here is that type inference can be reduced to efficient linear programming in the same way as for classic AARA for purely functional programs. This is also true for more complex polynomial type annotations [18].

The type inference works as follows. We first perform a standard, unification-based type inference for simple types. We then annotate the derivation tree with (yet unknown) potential annotations that will be determined by an LP solver. To generate the linear program, we apply the annotated type rules from Section 3 and collect the local constraints that need to hold for the annotations. We then minimize the initial potential using the LP solver.

To make the material more accessible, we have presented the potential annotations for references in the first-order setting. However, the proposed technique scales to higher-order programs and polynomial bounds [19].

Assume we would add references of ML type $T \equiv (T_1 \to T_2)$ *ref* to our language, which we can use to store first-order functions. In the annotated type system we would simply replace T with the annotated version $A \xrightarrow{q/q'} A$ *ref*. Like previous work [19], we can basically leave the rules L:Swap, L:Assign, L:DRef, and L:Ref unchanged; the same is true for the rules concerning array operations. This means that we fix a function annotation for each heap location that is referenced by a higher-order reference. Intuitively, every function that is stored in such a location would have to adhere to the resource behavior that is specified by the type. On the other hand, we can assume that a function has the specified worst-case behavior if we dereference a function value. Since we often need to use a function with different valid potential annotations, we implemented a generalization in RAML; functions are typed with a set of annotated function types $A \xrightarrow{q/q'} A$. However, the type rules remain conceptually identical.

Similarly, we do not have to alter the type rules for references and arrays when switching to univariate polynomial potential annotations [20]. In this scenario, we have potential annotations that are coefficients for more complex terms like $\binom{n}{k}$ rather than just the number of nodes n. Fortunately, `swap` does not alter the size of data structures, and therefore the rules do not have to be changed.

The situation is more complex when we consider multivariate polynomial potential annotations [18]. The difficulty is to decide how to handle mixed potential of the form $n{\cdot}m$ where m corresponds to the size of a list stored in a reference and n corresponds to the size of another data structure. Our current approach to this is to simply require that such a potential annotation is zero. This amounts to deriving multivariate bounds for data that is entirely stored within a single reference but not for data that is stored across multiple different references.

7 Related Work

Automatic amortized resource analysis (AARA) has been introduced to derive linear bounds on the number of heap allocations in a simple, strict, and first-order functional language [22]. Linear AARA has been extended to work with higher-order functions [27], object-oriented programs [23], lazy evaluation [37], imperative integer programs [9], pointer-based data structures [4], polynomial bounds [18], and term rewriting [25]. Many of the features for strict functional programs have been combined in Resource Aware ML [19]. In contrast to the presented technique, none of the aforementioned works allow the derivation of bounds that depend on the sizes of data structures that are stored in references and arrays.

Most closely related to our work is the treatment of references in recent work on RAML [21, 19]. Previous techniques allow references but statically ensure that the cost of computation does not depend on the sizes of data structures that have been retrieved from references and arrays. However, the higher-order system [19] derives bounds for programs whose cost depend on applications of functions that have been stored in and dereferenced from references and arrays. The innovation in this work is to allow the cost of computation to depend on data that is stored in references and arrays. There also exist AARAs for mutable heap data structures that are integrated in separation logic [4] and object-oriented type systems (in RAJA) [23, 26]. These type systems are somewhat incomparable but in many ways more expressive compared to the introduced system. The price they pay is that automatic bound

inference is challenging and often only possible when user annotations are provided. An advantage of our method is that it can naturally and automatically derive bounds that depend on the size of cyclic data structures, as demonstrated in Section 5.

There are also other type-based approaches to bound analysis. They are based on linear dependent types [28, 29] and type annotations [12, 31]. Cicek et al. [11] study a type system for incremental complexity. Moreover, there are analyses for functional programs that are based on solving and deriving (potentially higher-order) recurrence relations [6, 13]. None of these analysis systems can deal with side-effects and most have only basic support for automation.

Another research direction is to apply techniques from term rewriting to complexity analysis [33, 8, 5]; sometimes in combination with amortized analysis [24]. However, existing techniques seem to be restricted to purely functional programs and time complexity.

Approaches to resource analysis based on abstract interpretation [1, 3, 7, 10, 15, 16, 35] focus on bounds that depend on integers. There are techniques that take into account mutable heap structures by abstracting their sizes with ghost variables [2]. However, it is unclear how well these techniques scale to data with cycles, as well as with nested data structures that are potentially stored in arrays.

8 Conclusion

We have extended automatic amortized resource analysis to determine bounds for programs whose resource consumption depends on data stored in mutable heap cells. Moreover, we have followed the design philosophy of the ML type system in the sense that we refrain from tracking effects within the type system itself. In order to track resource usage that depends on the size of data stored in mutable cells, we require that the potential annotations for the types of these cells are invariant during the execution. We used a primitive `swap` operation that allows us to make use of the potential in references and arrays. The type rules ensure that we "swap in" data with the same potential annotation when we extract potential from a memory cell. To prove the non-trivial soundness theorem, we have used memory typing, which allows us to track relevant heap cells while not falling victim to aliasing.

Our analysis preserves the benefits of AARA such as compositionality and reduction of type inference to linear constraint solving. However, our work is also a departure from previous work in the sense that we extend the programming language with a new construct that guides the resource bound analysis instead of purely focusing on making the analysis work well for existing code. The additional burden that we put on the programmer is balanced by an elegant, clear, and transparent type system. A user of our system only needs to remember one simple rule: if the resource usage of the program depends on the size of dereferenced data, then the data has to be dereferenced using `swap`.

References

1 Elvira Albert, Puri Arenas, Samir Genaim, Germán Puebla, and Damiano Zanardini. Cost Analysis of Java Bytecode. In *16th Euro. Symp. on Prog. (ESOP'07)*, pages 157–172, 2007.

2 Elvira Albert, Richard Bubel, Samir Genaim, Reiner Hähnle, and Guillermo Román-Díez. Verified Resource Guarantees for Heap Manipulating Programs. In *15th Int. Conf. on Fundamental Approaches to Software Engineering (FASE'12)*, pages 130–145, 2012.

3 Elvira Albert, Jesús Correas Fernández, and Guillermo Román-Díez. Non-cumulative Resource Analysis. In *Tools and Algorithms for the Construction and Analysis of Systems – 21st Int. Conf., (TACAS'15)*, pages 85–100, 2015.

4 Robert Atkey. Amortised Resource Analysis with Separation Logic. In *19th Euro. Symp. on Prog. (ESOP'10)*, pages 85–103, 2010.

5 Martin Avanzini, Ugo Dal Lago, and Georg Moser. Analysing the Complexity of Functional Programs: Higher-Order Meets First-Order. In *29th Int. Conf. on Functional Programming (ICFP'15)*, 2012.

6 Ralph Benzinger. Automated Higher-Order Complexity Analysis. *Theor. Comput. Sci.*, 318(1-2):79–103, 2004.

7 Régis Blanc, Thomas A. Henzinger, Thibaud Hottelier, and Laura Kovács. ABC: Algebraic Bound Computation for Loops. In *Logic for Prog., AI., and Reasoning – 16th Int. Conf. (LPAR'10)*, pages 103–118, 2010.

8 Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and Jürgen Giesl. Alternating Runtime and Size Complexity Analysis of Integer Programs. In *Tools and Alg. for the Constr. and Anal. of Systems – 20th Int. Conf. (TACAS'14)*, pages 140–155, 2014.

9 Quentin Carbonneaux, Jan Hoffmann, and Zhong Shao. Compositional Certified Resource Bounds. In *36th Conference on Programming Language Design and Implementation (PLDI'15)*, 2015. Artifact submitted and approved.

10 Pavol Cerný, Thomas A. Henzinger, Laura Kovács, Arjun Radhakrishna, and Jakob Zwirchmayr. Segment Abstraction for Worst-Case Execution Time Analysis. In *24th European Symposium on Programming (ESOP'15)*, pages 105–131, 2015.

11 Ezgi Çiçek, Deepak Garg, and Umut A. Acar. Refinement Types for Incremental Computational Complexity. In *24th European Symposium on Programming (ESOP'15)*, pages 406–431, 2015.

12 Nils Anders Danielsson. Lightweight Semiformal Time Complexity Analysis for Purely Functional Data Structures. In *35th ACM Symp. on Principles Prog. Langs. (POPL'08)*, pages 133–144, 2008.

13 Norman Danner, Daniel R. Licata, and Ramyaa Ramyaa. Denotational Cost Semantics for Functional Languages with Inductive Types. In *29th Int. Conf. on Functional Programming (ICFP'15)*, 2012.

14 Antonio Flores-Montoya. Upper and Lower Amortized Cost Bounds of Programs Expressed as Cost Relations. In *Formal Methods – 21st International Symposium (FM'16)*, pages 254–273, 2016.

15 Antonio Flores-Montoya and Reiner Hähnle. Resource Analysis of Complex Programs with Cost Equations. In *Programming Languages and Systems – 12th Asian Symposiu (APLAS'14)*, pages 275–295, 2014.

16 Sumit Gulwani, Krishna K. Mehra, and Trishul M. Chilimbi. SPEED: Precise and Efficient Static Estimation of Program Computational Complexity. In *36th ACM Symp. on Principles of Prog. Langs. (POPL'09)*, pages 127–139, 2009.

17 Sumit Gulwani and Florian Zuleger. The Reachability-Bound Problem. In *Conf. on Prog. Lang. Design and Impl. (PLDI'10)*, pages 292–304, 2010.

18 Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Multivariate Amortized Resource Analysis. In *38th Symposium on Principles of Programming Languages (POPL'11)*, 2011.

19 Jan Hoffmann, Ankush Das, and Shu-Chun Weng. Towards Automatic Resource Bound Analysis for OCaml. In *44th Symposium on Principles of Programming Languages (POPL'17)*, 2017.

20 Jan Hoffmann and Martin Hofmann. Amortized Resource Analysis with Polynomial Potential. In *19th European Symposium on Programming (ESOP'10)*, 2010.

21 Jan Hoffmann and Zhong Shao. Type-Based Amortized Resource Analysis with Integers and Arrays. In *12th International Symposium on Functional and Logic Programming (FLOPS'14)*, 2014.

22 Martin Hofmann and Steffen Jost. Static Prediction of Heap Space Usage for First-Order Functional Programs. In *30th ACM Symp. on Principles of Prog. Langs. (POPL'03)*, pages 185–197, 2003.

23 Martin Hofmann and Steffen Jost. Type-Based Amortised Heap-Space Analysis. In *15th Euro. Symp. on Prog. (ESOP'06)*, pages 22–37, 2006. doi:10.1007/11693024_3.

24 Martin Hofmann and Georg Moser. Amortised Resource Analysis and Typed Polynomial Interpretations. In *Rewriting and Typed Lambda Calculi (RTA-TLCA;14)*, pages 272–286, 2014.

25 Martin Hofmann and Georg Moser. Multivariate Amortised Resource Analysis for Term Rewrite Systems. In *13th International Conference on Typed Lambda Calculi and Applications (TLCA'15)*, pages 241–256, 2015.

26 Martin Hofmann and Dulma Rodriguez. Efficient Type-Checking for Amortised Heap-Space Analysis. In *18th Conf. on Comp. Science Logic (CSL'09)*. LNCS, 2009.

27 Steffen Jost, Kevin Hammond, Hans-Wolfgang Loidl, and Martin Hofmann. Static Determination of Quantitative Resource Usage for Higher-Order Programs. In *37th ACM Symp. on Principles of Prog. Langs. (POPL'10)*, pages 223–236, 2010.

28 Ugo Dal Lago and Marco Gaboardi. Linear Dependent Types and Relative Completeness. In *26th IEEE Symp. on Logic in Computer Science (LICS'11)*, pages 133–142, 2011.

29 Ugo Dal Lago and Barbara Petit. The Geometry of Types. In *40th ACM Symp. on Principles Prog. Langs. (POPL'13)*, pages 167–178, 2013.

30 Benjamin Lichtman and Jan Hoffmann. Arrays and references in resource aware ml. Technical report, Carnegie Mellon University, 2017.

31 Ravichandhran Madhavan, Sumith Kulal, and Viktor Kuncak. Contract-based resource verification for higher-order functions with memoization. In *44th Symposium on Principles of Programming Languages POPL'17*, 2017.

32 Greg Morrisett, Amal Ahmed, and Matthew Fluet. L3: A linear language with locations. In *Typed Lambda Calculi and Applications: 7th International Conference, TLCA 2005, Nara, Japan, April 21-23, 2005. Proceedings*, pages 293–307, 2005.

33 Lars Noschinski, Fabian Emmes, and Jürgen Giesl. Analyzing Innermost Runtime Complexity of Term Rewriting by Dependency Pairs. *J. Autom. Reasoning*, 51(1):27–56, 2013.

34 Benjamin C. Pierce. *Advanced Topics in Types and Programming Languages*. The MIT Press, 2004.

35 Moritz Sinn, Florian Zuleger, and Helmut Veith. A Simple and Scalable Approach to Bound Analysis and Amortized Complexity Analysis. In *Computer Aided Verification – 26th Int. Conf. (CAV'14)*, pages 743–759, 2014.

36 Frederick Smith, David Walker, and J. Gregory Morrisett. Alias types. In *Proceedings of the 9th European Symposium on Programming Languages and Systems*, ESOP'00, pages 366–381, London, UK, UK, 2000. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=645394.651903.

37 Pedro B. Vasconcelos, Steffen Jost, Mário Florido, and Kevin Hammond. Type-Based Allocation Analysis for Co-recursion in Lazy Functional Languages. In *24th European Symposium on Programming (ESOP'15)*, pages 787–811, 2015.

A Rules Defining the Big-Step Operational Semantics

We use $\overline{n}$ to denote the unary representation of the number $n \in \mathbb{N}$ (as defined in Section 3) for use in the rules for array operations. For example, $\overline{0} = \text{zero}$, $\overline{1} = \text{succ(zero)}$, and so on.

$$\frac{V(x) = \ell}{F, V, H_M \vdash x \Downarrow (\ell, H) \mid M^{\text{var}}}\ (\text{E:Var}) \qquad \frac{}{F, V, H_M \vdash () \Downarrow (\text{Null}, H) \mid M^{\text{triv}}}\ (\text{E:Triv})$$

$$\frac{}{F, V, H_M \vdash \text{true} \Downarrow (\texttt{tt}, H) \mid M^{\text{true}}}\ (\text{E:True}) \qquad \frac{}{F, V, H_M \vdash \text{false} \Downarrow (\texttt{ff}, H) \mid M^{\text{false}}}\ (\text{E:False})$$

$$\frac{(f, \vec{y}, e_f) \in F \qquad F, [y_1 \mapsto V(x_1), \cdots, y_n \mapsto V(x_n)], H_M \vdash e_f \Downarrow (v, H') \mid (q, q')}{F, V, H_M \vdash f(x_1, \ldots, x_n) \Downarrow (v, H') \mid M_1^{\text{fun}_n} \cdot (q, q') \cdot M_2^{\text{fun}_n}}\ (\text{E:Fun})$$

$$\frac{F, V, H_M \vdash e_1 \Downarrow (v_1, H_1) \mid (q, q') \qquad F, V[x \mapsto v_1], H_1 M \vdash e_2 \Downarrow (v_2, H_2) \mid (p, p')}{F, V, H_M \vdash \text{let } x = e_1 \text{ in } e_2 \Downarrow (v_2, H_2) \mid M_1^{\text{let}} \cdot (q, q') \cdot M_2^{\text{let}} \cdot (p, p') \cdot M_3^{\text{let}}}\ (\text{E:Let})$$

$$\frac{c_i \in \text{CID} \qquad v = (\mathsf{constr}_{c_i}, V(x_1), \ldots, V(x_k)) \qquad H' = H, \ell \mapsto v \qquad \ell \notin \text{dom}(H)}{F, V, H_M \vdash c_i \langle x_1, \ldots, x_k \rangle \Downarrow (\ell, H') \mid M^{\text{cons}}}\ (\text{E:Constr})$$

$$\frac{H(V(x)) = (\mathsf{constr}_{c_i}, v_1, \ldots, v_k) \qquad F, V[x_1 \mapsto v_1, \ldots, x_k \mapsto v_k], H_M \vdash e_1 \Downarrow (v, H') \mid (q, q')}{F, V, H_M \vdash \text{match } x \text{ with } c_i \langle x_1, \ldots, x_k \rangle \Rightarrow e_1 \mid e_2 \Downarrow (v, H') \mid M_1^{\text{matT}} \cdot (q, q') \cdot M_2^{\text{matT}}}\ (\text{E:Mat1})$$

$$\frac{H(V(x)) \neq (\mathsf{constr}_{c_i}, v_1, \ldots, v_k) \qquad F, V, H_M \vdash e_2 \Downarrow (v, H') \mid (q, q')}{F, V, H_M \vdash \text{match } x \text{ with } c_i \langle x_1, \ldots, x_k \rangle \Rightarrow e_1 \mid e_2 \Downarrow (v, H') \mid M_1^{\text{matF}} \cdot (q, q') \cdot M_2^{\text{matF}}}\ (\text{E:Mat2})$$

$$\frac{V(x) = v' \qquad V' = V \setminus x \qquad F, V'[x_1 \mapsto v', x_2 \mapsto v'], H_M \vdash e \Downarrow (v, H') \mid (q, q')}{F, V, H_M \vdash \text{share } x \text{ as } (x_1, x_2) \text{ in } e \Downarrow (v, H') \mid (q, q')}\ (\text{E:Share})$$

$$\frac{H' = H, \ell \mapsto V(x) \qquad \ell \notin \text{dom}(H)}{F, V, H_M \vdash \text{ ref } x \Downarrow (\ell, H') \mid M^{\text{ref}}}\ (\text{E:Ref}) \qquad \frac{\ell = H(V(x_1)) \qquad H' = H[V(x_1) \mapsto V(x_2)]}{F, V, H_M \vdash \text{swap}(x_1, x_2) \Downarrow (\ell, H') \mid M^{\text{swap}}}\ (\text{E:Swap})$$

$$\frac{\ell = H(V(x))}{F, V, H_M \vdash\ !x \Downarrow (\ell, H) \mid M^{\text{dref}}}\ (\text{E:DRef}) \qquad \frac{H' = H[V(x_1) \mapsto V(x_2)]}{F, V, H_M \vdash x_1 := x_2 \Downarrow (\text{Null}, H) \mid M^{\text{assign}}}\ (\text{E:Assign})$$

$$\frac{\begin{array}{c} H(V(x)) = \overline{n} \qquad F, V, H_M \vdash e \Downarrow (v, H') \mid (q, q') \\ H'' = H', \ell \mapsto (\sigma, n), \ell_1 \mapsto v, \ldots, \ell_n \mapsto v \qquad \forall i : \sigma(i) = \ell_{i+1} \qquad \ell, \ell_1, \ldots, \ell_n \notin \text{dom}(H) \end{array}}{F, V, H_M \vdash \text{create}(x, e) \Downarrow (\ell, H'') \mid M^{\text{create}} \cdot (q, q')^n}\ (\text{E:Create})$$

$$\frac{H(V(x_1)) = (\sigma, n) \qquad H(V(x_2)) = \overline{i} \qquad 0 \leq i < n}{F, V, H_M \vdash \text{get}(x_1, x_2) \Downarrow (H(\sigma(i)), H) \mid M^{\text{get}}}\ (\text{E:Get})$$

$$\frac{H(V(x_1)) = (\sigma, n) \qquad H(V(x_2)) = \overline{i} \qquad 0 \leq i < n \qquad H' = H[\sigma(i) \mapsto V(x_3)]}{F, V, H_M \vdash \text{set}(x_1, x_2, x_3) \Downarrow (\text{Null}, H') \mid M^{\text{set}}}\ (\text{E:Set})$$

$$\frac{\begin{array}{c} H(V(x_1)) = (\sigma, n) \qquad H(V(x_2)) = \overline{i} \qquad 0 \leq i < n \\ v = H(\sigma(i)) \qquad H' = H[\sigma(i) \mapsto V(x_3)] \end{array}}{F, V, H_M \vdash \text{aswap}(x_1, x_2, x_3) \Downarrow (v, H') \mid M^{\text{aswap}}}\ (\text{E:Aswap})$$

$$\frac{H(V(x)) = (\sigma, n) \qquad H' = H, \ell \mapsto \overline{n} \qquad \ell \notin \text{dom}(H)}{F, V, H_M \vdash \text{length}(x) \Downarrow (\ell, H') \mid M^{\text{len}}}\ (\text{E:Len})$$

B Rules Defining the Simple Typing Judgement

$$\frac{}{\Sigma;\emptyset \vdash \text{true} : B}\ \text{(T:True)} \qquad \frac{}{\Sigma;\emptyset \vdash \text{false} : B}\ \text{(T:False)}$$

$$\frac{}{\Sigma;\emptyset \vdash () : 1}\ \text{(T:Triv)} \qquad \frac{}{\Sigma;x{:}T \vdash x : T}\ \text{(T:Var)}$$

$$\frac{n \in \mathbb{N}}{\Sigma;\emptyset \vdash n : N}\ \text{(T:Nat)} \qquad \frac{\Sigma(f) = (T_1,\dots,T_n) \to T}{\Sigma;\Gamma,x_1{:}T_1,\dots,x_n{:}T_n \vdash f(x_1,\dots,x_n) : T}\ \text{(T:Fun)}$$

$$\frac{\Sigma;\Gamma_1 \vdash e_1 : T' \qquad \Sigma;\Gamma_2,x{:}T' \vdash e_2 : T}{\Sigma;\Gamma_1,\Gamma_2 \vdash \text{let } x = e_1 \text{ in } e_2 : T}\ \text{(T:Let)}$$

$$\frac{T = \mu X.\{\cdots \mid c_i{:}(T_1',\dots,T_k') \mid \cdots\} \qquad \begin{array}{c} c \in \text{CID} \\ \forall 1 \le j \le k.\, T_j = T_j' \vee (T_j = T \wedge T_j' = X) \end{array}}{\Sigma;x_1{:}T_1,\dots,x_k{:}T_k \vdash c_i\ (x_1,\dots,x_k) : T}\ \text{(T:Constr)}$$

$$\frac{\begin{array}{c} c \in \text{CID} \qquad T' = \mu X.\{\cdots \mid c_i{:}(T_1',\dots,T_k') \mid \cdots\} \\ \Sigma;\Gamma,x_1{:}[T'/X]T_1',\dots,x_k{:}[T'/X]T_k' \vdash e_1 : T \qquad \Sigma;\Gamma,x{:}T' \vdash e_2 : T \end{array}}{\Sigma;\Gamma,x : T' \vdash \text{match } x \text{ with } c_i\ \langle x_1,\dots,x_k\rangle \Rightarrow e_1 \mid e_2 : T}\ \text{(T:Mat)}$$

$$\frac{\Sigma;\Gamma \vdash e : T}{\Sigma;\Gamma,x{:}T' \vdash e : T}\ \text{(T:Weaken)} \qquad \frac{\Sigma;\Gamma,x_1{:}T',x_2{:}T' \vdash e : T}{\Sigma;\Gamma,x{:}T' \vdash \text{share } x \text{ as } (x_1,x_2) \text{ in } e : T}\ \text{(T:Share)}$$

$$\frac{}{\Sigma;x{:}T \vdash \text{ref } x : T \text{ ref}}\ \text{(T:Ref)} \qquad \frac{}{\Sigma;x{:}T \text{ ref} \vdash !x : T}\ \text{(T:DRef)}$$

$$\frac{}{\Sigma;x_1{:}T \text{ ref},x_2{:}T \vdash x_1 := x_2 : 1}\ \text{(T:Assign)}$$

$$\frac{}{\Sigma;x_1{:}T \text{ ref},x_2{:}T \vdash \text{swap}(x_1,x_2) : T}\ \text{(T:Swap)}$$

$$\frac{}{\Sigma;x_1{:}N,x_2{:}T \vdash \text{create}(x_1,x_2) : T \text{ array}}\ \text{(T:Create)}$$

$$\frac{}{\Sigma;x_1{:}T \text{ array},x_2{:}N \vdash \text{get}(x_1,x_2) : T}\ \text{(T:Get)}$$

$$\frac{}{\Sigma;x_1{:}T \text{ array},x_2{:}N,x_3{:}T \vdash \text{set}(x_1,x_2,x_3) : 1}\ \text{(T:Set)}$$

$$\frac{}{\Sigma;x_1{:}T \text{ array},x_2{:}N,x_3{:}T \vdash \text{aswap}(x_1,x_2,x_3) : T}\ \text{(T:Aswap)}$$

$$\frac{}{\Sigma;x{:}T \text{ array} \vdash \text{length}(x) : N}\ \text{(T:Len)}$$

C Rules Defining the Annotated Typing Judgement

$$\frac{}{\Sigma; \emptyset \left|^{q+M^{\text{true}}}_{q}\right. \text{true} : B}\ (\text{L:True}) \qquad \frac{}{\Sigma; \emptyset \left|^{q+M^{\text{false}}}_{q}\right. \text{false} : B}\ (\text{L:False}) \qquad \frac{}{\Sigma; \emptyset \left|^{q+M^{\text{triv}}}_{q}\right. () : 1}\ (\text{L:Triv})$$

$$\frac{}{\Sigma; x{:}A \left|^{q+M^{\text{var}}}_{q}\right. x : A}\ (\text{L:Var}) \qquad \frac{(A_1, \ldots, A_n) \xrightarrow{q/q'} A \in \Sigma(f)}{\Sigma; x_1{:}A_1, \ldots, x_n{:}A_n \left|^{q+M_1^{\text{fun}_n}}_{q'-M_2^{\text{fun}_n}}\right. f(x_1, \ldots, x_n) : A}\ (\text{L:Fun})$$

$$\frac{}{\Sigma; \emptyset \left|^{q+M^{\text{nat}}}_{q}\right. n : N}\ (\text{L:Nat}) \qquad \frac{\Sigma; \Gamma_1 \left|^{q-M_1^{\text{let}}}_{p}\right. e_1 : A' \qquad \Sigma; \Gamma_2, x{:}A' \left|^{p-M_2^{\text{let}}}_{q'+M_3^{\text{let}}}\right. e_2 : A}{\Sigma; \Gamma_1, \Gamma_2 \left|^{q}_{q'}\right. \text{let } x = e_1 \text{ in } e_2 : A}\ (\text{L:Let})$$

$$\frac{c \in \text{CID} \qquad A = \mu X^P.\{\cdots \mid c_i{:}(A_1', \ldots, A_k') \mid \cdots\} \qquad \forall 1 \le j \le k.\, A_j = A_j' \vee (A_j = A \wedge A_j' = X)}{\Sigma; x_1{:}A_1, \ldots, x_k{:}A_k \left|^{q+P[i]+M^{\text{cons}}}_{q}\right. c_i\ (x_1, \ldots, x_k) : A}\ (\text{L:Constr})$$

$$\frac{\begin{array}{c}c \in \text{CID} \qquad A' = \mu X^P.\{\cdots \mid c_i{:}(A_1', \ldots, A_k') \mid \cdots\} \\ \Sigma; \Gamma, x_1{:}[A'/X]A_1', \ldots, x_k{:}[A'/X]A_k' \left|^{q+P[i]-M_1^{\text{matT}}}_{q'+M_2^{\text{MatT}}}\right. e_1 : A \qquad \Sigma; \Gamma, x{:}A' \left|^{q-M_1^{\text{matF}}}_{q'+M_2^{\text{matF}}}\right. e_2 : A\end{array}}{\Sigma; \Gamma, x : A' \left|^{q}_{q'}\right. \text{match } x \text{ with } c_i\ \langle x_1, \ldots, x_k\rangle \Rightarrow e_1 \mid e_2 : A}\ (\text{L:Mat})$$

$$\frac{\Sigma; \Gamma \left|^{q}_{q'}\right. e : A}{\Sigma; \Gamma, x{:}A' \left|^{q}_{q'}\right. e : A}\ (\text{L:Weaken}) \qquad \frac{\Sigma; \Gamma, x_1{:}A_1, x_2{:}A_2 \left|^{q}_{q'}\right. e : A \qquad A' \curlyvee (A_1, A_2)}{\Sigma; \Gamma, x{:}A' \left|^{q}_{q'}\right. \text{share } x \text{ as } (x_1, x_2) \text{ in } e : A}\ (\text{L:Share})$$

$$\frac{\Sigma; \Gamma \left|^{p}_{p'}\right. e : A \qquad q \ge p \qquad q - p \ge q' - p'}{\Sigma; \Gamma \left|^{q}_{q'}\right. e : A}\ (\text{L:Relax}) \qquad \frac{}{\Sigma; x{:}A \left|^{q+M^{\text{ref}}}_{q}\right. \text{ref } x : A \text{ ref}}\ (\text{L:Ref})$$

$$\frac{|A'| = |A| \qquad A \curlyvee (A, A)}{\Sigma; x{:}A' \text{ ref} \left|^{q+M^{\text{dref}}}_{q}\right. !x : A}\ (\text{L:DRef}) \qquad \frac{}{\Sigma; \Gamma, x_1{:}A \text{ ref}, x_2{:}A \left|^{q+M^{\text{assign}}}_{q}\right. x_1 := x_2 : 1}\ (\text{L:Assign})$$

$$\frac{}{\Sigma; x{:}A \text{ array}^p \left|^{q+M^{\text{len}}}_{q}\right. \text{length}(x) : N^p}\ (\text{L:Len}) \qquad \frac{}{\Sigma; x_1{:}A \text{ ref}, x_2{:}A \left|^{q+M^{\text{swap}}}_{q}\right. \text{swap}(x_1, x_2) : A}\ (\text{L:Swap})$$

$$\frac{\Sigma; \cdot \left|^{p-r}_{0}\right. e : A}{\Sigma; x{:}N^p \left|^{q+M^{\text{create}}}_{q}\right. \text{create}(x, e) : A \text{ array}^r}\ (\text{L:Create})$$

$$\frac{|A'| = |A| \qquad A \curlyvee (A, A)}{\Sigma; x_1{:}A' \text{ array}^p, x_2{:}N^r \left|^{q+M^{\text{get}}}_{q}\right. \text{get}(x_1, x_2) : A}\ (\text{L:Get})$$

$$\frac{}{\Sigma; x_1{:}A \text{ array}^p, x_2{:}N^r, x_3{:}A \left|^{q+M^{\text{set}}}_{q}\right. \text{set}(x_1, x_2, x_3) : 1}\ (\text{L:Set})$$

$$\frac{}{\Sigma; x_1{:}A \text{ array}^p, x_2{:}N^r, x_3{:}A \left|^{q+M^{\text{aswap}}}_{q}\right. \text{aswap}(x_1, x_2, x_3) : A}\ (\text{L:Aswap})$$

Negative Translations and Normal Modality

Tadeusz Litak*[1], Miriam Polzer[2], and Ulrich Rabenstein[3]

1 **Informatik 8, FAU Erlangen-Nürnberg, Erlangen, Germany**
2 **Informatik 8, FAU Erlangen-Nürnberg, Erlangen, Germany**
3 **Informatik 8, FAU Erlangen-Nürnberg, Erlangen, Germany**

Abstract

We discuss the behaviour of variants of the standard negative translations – Kolmogorov, Gödel-Gentzen, Kuroda and Glivenko – in propositional logics with a unary normal modality. More specifically, we address the question whether negative translations as a rule embed faithfully a classical modal logic into its intuitionistic counterpart. As it turns out, even the Kolmogorov translation can go wrong with rather natural modal principles. Nevertheless, we isolate sufficient syntactic criteria ensuring adequacy of well-behaved (or, in our terminology, *regular*) translations for logics axiomatized with formulas satisfying these criteria, which we call *enveloped implications*. Furthermore, a large class of computationally relevant modal logics – namely, logics of type inhabitation for *applicative functors* (a.k.a. *idioms*) – turns out to validate the modal counterpart of the Double Negation Shift, thus ensuring adequacy of even the Glivenko translation. All our positive results are proved purely syntactically, using the minimal natural deduction system of Bellin, de Paiva and Ritter extended with additional axioms/combinators and hence also allow a direct formalization in a proof assistant (in our case Coq).

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases negative translations, intuitionistic modal logic, normal modality, double negation

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.27

1 Introduction

A normal modality $\Box$ over the intuitionistic propositional calculus (IPC) can be seen as a restricted universal quantifier, but also – given the interpretation of IPC in (bi)cartesian closed categories – as the syntactic counterpart of an endofunctor (monoidal w.r.t. the cartesian structure [6, 16]) or – from the Curry-Howard point of view – a type constructor distributing over conjunctions/products (see [6, 44, 31, 16, 36] for an overview and more references). It is, in short, the most natural way of extending IPC, which stops short of the full complexity of proper quantifiers (dependent types), i.e., the intuitionistic predicate calculus (IQC). The computational interpretation and importance of *negative translations* is well-known (cf., e.g., [27, 39, 1] and [48, Ch. 6]) and their behaviour is rather well-understood not only for IPC, but also for IQC. Modal logic also has a very well-developed metatheory, not only over the classical propositional calculus (CPC), but also over IPC. See, e.g., [26, 11, 49, 56, 57, 59].

It would seem then that the following problem should have been fully solved by now. Consider the simplest possible modal extension $\mathcal{L}_\Box$ of the syntax of IPC. Take the standard notion of a (*normal*) *logic* $\mathrm{i}_\Box\mathcal{Z}$ in such a language (§ 2) and whichever variant t of the standard negative translations the reader prefers (§ 3). Is such a translation t by default *adequate* for every $\mathrm{i}_\Box\mathcal{Z}$ – that is, given any $\phi \in \mathcal{L}_\Box$, is it the case that $\phi \in \mathrm{c}_\Box\mathcal{Z} = \mathrm{i}_\Box\mathcal{Z} + \mathsf{CPC}$

* Corresponding author: `https://www8.cs.fau.de/staff/litak/`.

iff $\phi^t \in \mathsf{i}_\Box\mathcal{Z}$ (§ 4)? If not, is it possible to find some general criteria on the axiomatization of $\mathsf{i}_\Box\mathcal{Z}$ which ensure adequacy – and show correctness of these criteria in a purely syntactic way (§ 5)? Can it happen that for entire classes of logic with good computational, type-theoretic and categorical motivation, the simplest possible translation (Glivenko) would be adequate (§ 6)?

And yet, to the best of our knowledge, while the question of adequacy of negative translations (we call this property $\neg\neg$-*completeness*) has been addressed for the minimal system $\mathsf{i}_\Box\mathsf{K}$ [11, 28] (see also § 7 for a discussion of [7]), there has been no systematic study for arbitrary axioms. Our paper aims to fill this gap.

Negative translations (mostly Glivenko) have been studied for other non-classical logics, especially substructural ones [23, 43, 20]. The Gödel-Gentzen translation also plays an important rôle in the recently developed theory of *possibility semantics* for modal logic [53, 28]. Moreover, our lack of systematic knowledge regarding $\neg\neg$-completeness contrasts with the existence of exhaustive studies of Gödel-McKinsey-Tarski-type translations of $\mathcal{L}_\Box$-logics [11, 56, 57].

On the other hand, negative translations can quickly go very wrong with natural and important extensions of intuitionistic logic. As we are going to discuss in § 4.1, a striking example is provided by the logic of bunched implications BI. Thus, beginning the systematic study of modal negative translations with a single normal $\Box$ seems the right level of generality at the moment.

A systematic investigation of $\neg\neg$-completeness seems warranted by Simpson's [46, Ch 3.2] influential *requirements* that an intuitionistic modal logic is supposed to satisfy, in particular, Requirement 3: *The addition of $\alpha \vee \neg\alpha$ to an intuitionistic modal logic should yield a standard classical modal logic.* This requirement turns out to be disputable from at least two points of view. On the one hand, it is too restrictive: there are entire classes of important intuitionistic modal logics which classically collapse to rather trivial systems. See the discussion of $\mathsf{i}_\Box\mathsf{R}$ and its extensions like PLL and SL in § 6, especially Remark 25 (ironically, even the Glivenko translation is adequate for such logics); other "classically near-degenerate" examples can be found in provability logics of extensions of HA, cf., e.g., [29, 55]. On the other hand, even if $\mathcal{Z}$ consists entirely of "standard" axioms, something is wrong with the relationship between $\mathsf{i}_\Box\mathcal{Z}$ and $\mathsf{c}_\Box\mathcal{Z}$ if negative translations are not adequate – and as it turns out, this can indeed happen with very simple *modal reduction principles* [58, §4.5]. Contrast for example $\mathsf{4}$, i.e., the transitivity axiom $\Box p \to \Box\Box p$, with axioms like $\mathsf{C4} : \Box\Box p \to \Box p$ or $\flat\mathsf{4} : \Box\Box p \to \Box\Box\Box p$. In the light of our Enveloped Implication Theorem, i.e., Theorem 18, the logic $\mathsf{i}_\Box\mathsf{4}$ is $\neg\neg$-complete (Corollary 19), but $\mathsf{i}_\Box\mathsf{C4}$ (Example 11 and Theorem 12) or $\mathsf{i}_\Box\flat\mathsf{4}$ (Example 13 and Theorem 14) are not.

An interesting feature of general positive results in the present paper is that we were able to show them purely syntactically. This contrasts with less constructive methods typically employed in *modal logic as "die Klassentheorie"* [58], cf. § 2 for a discussion of these points and more details. Note that a similar approach was pursued in investigation of the Glivenko translation for substructural logics by Ono and coauthors [43, 20]. This allows a straightforward formalization in a proof assistant, to which the reader is referred for omitted details of proofs.[1] It is possible, however, to provide semantic characterizations of $\neg\neg$-completeness: this is briefly discussed in § 7 and the details are postponed to future work.

[1] Plain Coq in our case, but translating this development to any other setting would be straightforward: we only used a small number of our own tacticals to shorten proof scripts. To download, `git clone git://git8.cs.fau.de/dnegmod`. Web front-end available at `https://cal8.cs.fau.de/redmine/projects/dnegmod`. As described in the README file, full documentation can be produced by `make all-gal.pdf`. We have tested the development in several versions of Coq, ranging from 8.4pl4 (November 2015) to 8.6 (May 2017).

2 Logics and proof systems

Modal formulas over a supply of propositional variables Σ (unless stated otherwise, fixed and dropped from the notation) are defined by

$$\mathcal{L}_{\Box}\Sigma \quad \phi,\psi ::= \bot \mid p \mid \phi \to \psi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \Box\phi$$

where $p \in \Sigma$. $\mathcal{L}$ denotes the pure propositional language (i.e., without $\Box$). We follow standard conventions regarding binding priorities and associativity of connectives; in particular, we assume $\to$ associates to the right. As usual, $\neg\phi$ is $\phi \to \bot$. To save space and improve readability, let us slightly tweak typographically $\neg\neg\phi$ to $\neg\!\neg\phi$.[2] Also, as usual, a substitution is a function $s : \Sigma \mapsto \mathcal{L}_{\Box}$. Its uniquely determined extension $\overline{s} : \mathcal{L}_{\Box} \mapsto \mathcal{L}_{\Box}$ will be notationally conflated with s itself. A particularly important substitution $s_{\neg\neg}(\cdot)$ is generated by $s_{\neg\neg}(p) := \neg\!\neg p$ for each $p \in \Sigma$.

Unlike the classical case, it matters that $\Box$ is primitive and $\Diamond$ is not. There are several possible approaches to combining $\Box$ and $\Diamond$ in the intuitionistic setting (overviews can be found, e.g., in [11, 46, 56, 57]); to save space and promote clarity, we leave the discussion of negative translations in each of these setups to future work.

2.1 Intuitionistic modal logics à la Hilbert

As usual, by a(*n intuitionistic normal modal*) *logic* we understand a set of formulas closed under

- the axioms of intuitionistic propositional calculus (IPC),
- K: $\Box(p \to q) \to \Box p \to \Box q$,
- Modus Ponens: $\dfrac{\phi \to \psi \qquad \phi}{\psi}$,
- the rule of *Necessitation*, also known as the *Gödel rule*: $\dfrac{\phi}{\Box\phi}$,
- and, finally, *substitution*: $\dfrac{\phi}{s(\phi)}$ for any s.

The smallest set of formulas closed under these rules is denoted as $\mathsf{i}_{\Box}\mathsf{K}$. For any $\mathcal{Z} \subseteq \mathcal{L}_{\Box}$, the smallest intuitionistic normal modal logic containing $\mathcal{Z}$ – i.e., the normal extension of $\mathsf{i}_{\Box}\mathsf{K}$ axiomatized by $\mathcal{Z}$ – is denoted by $\mathsf{i}_{\Box}\mathcal{Z}$.[3] Furthermore, $\mathsf{c}_{\Box}\mathcal{Z}$ is the smallest logic (in the above sense) containing $\mathcal{Z}$ and the classical propositional calculus (CPC); we can formally define it as $\mathsf{i}_{\Box}\mathcal{Z} + \neg\!\neg p \to p$. Note here that for sets of axioms, it is notationally convenient to omit singleton brackets and replace $\cup$ by $+$. The smallest modal logic containing CPC is denoted as $\mathsf{c}_{\Box}\mathsf{K}$. Finally, let us write $\phi \vdash_{\mathsf{i}_{\Box}\mathcal{Z}} \psi$ for $\phi \to \psi \in \mathsf{i}_{\Box}\mathcal{Z}$ and $\phi \dashv\vdash_{\mathsf{i}_{\Box}\mathcal{Z}} \psi$ for $\phi \leftrightarrow \psi \in \mathsf{i}_{\Box}\mathcal{Z}$.

Such a Hilbert-style definition of a logic is rather inconvenient for proof search or, indeed, for most proofs by structural induction on formulas. Nevertheless, it lends itself to a natural process of *algebraization*, Lindenbaum-Tarski style: in the case of intuitionistic (or classical) $\Box$ one obtains expansions of Heyting (or Boolean) algebras with equational axioms capturing that $\Box$ distributes over arbitrary finite (possibly empty) conjunctions. Additional equations

[2] This notation indicates the perspective on double negation as a modality, especially over negation-free and bottom-free intutionistic syntax [10, 17]. We will discuss this briefly in § 7.

[3] Conceivable notational conventions to be found in the literature include also, e.g., $\mathsf{iK}_{\Box} + \mathcal{Z}$ or $\mathsf{IntK}_{\Box} \oplus \mathcal{Z}$.

encode additional axioms from $\mathcal{Z}$. Futhermore, such algebras can be dually represented as *Esakia spaces*, *Priestley spaces*, *descriptive general frames* or (in the classical case) *Stone coalgebras for the Vietoris functor* [32], which allows topological or model-theoretic technology in investigating large classes of logics. The use of algebras and/or their duals and the need to formalize a reasonably large fragment of their metatheory, however, complicates formalization in proof assistants, limiting their transparency, transferability and the potential for program extraction and computational interpretation. It is also worth adding that most standard approaches to this line of investigation are rather non-constructive; even in purely algebraic research, one often relies on various consequences of the Axiom of Choice (such as the existence of a subdirect decomposition of any algebra) and still more so in representation and duality results leading to classes of frames, spaces and coalgebras mentioned above.[4]

2.2 Intuitionistic modal logics à la Gentzen

For logics with additional axioms of a specific syntactic shape, there are generic methods producing, e.g., suitable hyper-sequent calculi allowing cut-elimination. Especially in the substructural setting, the scope of such methods is systematically studied by *algebraic* or *systematic proof theory* (cf., e.g., [14, 15]), but this line of work also reveals that there are fundamental limitations on the shape of suitable axioms. Similar restrictions apply to labelled natural-deduction calculi for extensions of $\mathsf{i}_{\Box}\mathsf{K}$ (cf., e.g., [46, §6.3]). And, needless to say, there are many examples of ostentatiously misbehaving formulas, in particular those axiomatizing undecidable logics.

Is there any chance then of using Gentzen-inspired methods to prove general characterization results for *arbitrary* axioms such as our Theorems 15 and 18 below?

The solution turns out to be surprisingly simple: we do not need to postulate unrestricted substitution as an explicit rule. As first observed, to the best of our knowledge, by Sobociński (in a Hilbert-style setting) [47, 34], propositional deductions can be put in a form where the substitution rule is *only applied to axioms*. This idea can be replayed in a Gentzen-style setup. Obviously, we cannot assume that such systems with additional axioms would allow in general results like normalization/cut-elimination, not to mention Martin-Löf-style *local soundness* and *local completeness* [44]. But they will do a perfectly fine job in improving the support for structural induction on formulas, and this is all we need in this paper.

Given the simplicity of our goals, there is no need to complicate our presentation (and the associated Coq formalization) with additional apparatus like labels, multiple contexts, nested sequents, hypersequents etc. Instead, just like in the work of Kakutani [30], we take as our base the standard, single-context ND-calculus for $\mathsf{i}_{\Box}\mathsf{K}$ by Bellin, de Paiva and Ritter [5, 6, 16] – with additional tweaks such as not only presenting everything in a sequent notation, but explicitly using multisets. Essentially, this yields a calculus equivalent to those obeying the so-called *Complete Discharge Convention* [50],[51, §2.1.10].

Thus, let $\Gamma, \Gamma', \Delta \ldots$ range over finite multisets of formulas from $\mathcal{L}_{\Box}$. As usual, let $\bigwedge\Gamma$ be the conjunction of all formulas from Γ. For any given set of axioms $\mathcal{Z}$, the rules of its corresponding calculus $\mathsf{Ni}_{\Box}\mathcal{Z}$ governing derivability of judgements of the form $\Gamma \Rightarrow \phi$ are given in Figure 1. In proofs, one can freely use rules derivable in $\mathsf{Ni}_{\Box}\mathsf{K}$, such as

$$\neg\neg\to_1 \frac{\vdash_{\mathsf{Ni}_{\Box}\mathcal{Z}} \Rightarrow \phi \to \psi}{\vdash_{\mathsf{Ni}_{\Box}\mathcal{Z}} \Rightarrow \neg\neg\phi \to \neg\neg\psi} \qquad \neg\neg\to_2 \frac{\vdash_{\mathsf{Ni}_{\Box}\mathcal{Z}} \Rightarrow \phi \to \psi}{\vdash_{\mathsf{Ni}_{\Box}\mathcal{Z}} \Rightarrow \neg\neg(\neg\neg\phi \to \neg\neg\psi)}$$

[4] There are more constructive approaches, but they are not often used by modal logicians studying entire lattices of logics; thus, their metatheory and the body of relevant results on offer are less developed.

Intuitionistic propositional rules:

$$\textsc{In}\ \frac{}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \phi, \Gamma \Rightarrow \phi} \qquad \top_I\ \frac{}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \top} \qquad \bot_E\ \frac{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \bot}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi}$$

$$\to_I\ \frac{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma, \phi \Rightarrow \psi}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi \to \psi} \qquad \to_E\ \frac{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi \to \psi \qquad \vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \psi}$$

$$\wedge_I\ \frac{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi \qquad \vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \psi}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi \wedge \psi} \qquad \wedge_{E1}\ \frac{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi \wedge \psi}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi} \qquad \wedge_{E2}\ \frac{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi \wedge \psi}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \psi}$$

$$\vee_{I1}\ \frac{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi \vee \psi} \qquad \vee_{I2}\ \frac{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \psi}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi \vee \psi}$$

$$\vee_E\ \frac{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi \vee \psi \qquad \vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \phi, \Gamma \Rightarrow \chi \qquad \vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \psi, \Gamma \Rightarrow \chi}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \chi}$$

The Bellin, de Paiva and Ritter rule for $\mathsf{i}_\Box\mathsf{K}$ [5, 6, 30, 16]:

$$\Box_\mathsf{K}\ \frac{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \Box\phi_1 \qquad \ldots \vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \Box\phi_n \qquad \vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \phi_1, \ldots, \phi_n \Rightarrow \psi}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \Box\psi}$$

The Sobociński-style rule for additional axioms [47, 34]:

$$\textsc{AxSb}\ \frac{\zeta \in \mathcal{Z} \qquad s \text{ a substitution}}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow s(\zeta)}$$

Figure 1 Rules of $\mathsf{Ni}_\Box\mathcal{Z}$.

▶ **Theorem 1.** *For any $\mathcal{Z}$, Γ and α, $\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \alpha$ iff $\bigwedge\Gamma \to \alpha \in \mathsf{i}_\Box\mathcal{Z}$.*

2.3 Semantics

We keep discussion of semantics to a minimum. We only need intuitionistic Kripke frames to disprove the validity of certain formulas. We refer the reader to numerous references [26, 11, 46, 36] for a more detailed discussion. Thus, recall that a(n *intuitionistic* $\Box$-) *frame* is of the form $\mathfrak{F} := \langle W, \uparrow, \rightsquigarrow\rangle$, where $\uparrow \subseteq W\times W$ is a partial order on W and $\rightsquigarrow \subseteq W\times W$ satifies $\uparrow;\rightsquigarrow;\uparrow \subseteq \rightsquigarrow$, where ";" is the relational composition; in other words, $\rightsquigarrow$ is closed under pre- and postfixing with $\uparrow$. A valuation V maps elements of Σ to subsets of W, which are upward closed w.r.t. $\uparrow$. It is used to define a forcing relation $\Vdash$ between elements of W and formulas of $\mathcal{L}_\Box$ using the usual clauses for intuitionistic connectives with $\uparrow$ used to interpret implication and with $\rightsquigarrow$ used to interpret $\Box$, i.e.,

$\mathfrak{F}, V, w \Vdash \Box\phi$ if for any w' s.t. $w \rightsquigarrow w'$, we have that $\mathfrak{F}, V, w' \Vdash \phi$.

The interaction condition ensures that denotations of all formulas are upward closed w.r.t. $\uparrow$; in fact, even a much weaker one would do, but as long as normal $\Box$ is the only additional modality, imposing $\uparrow;\rightsquigarrow;\uparrow \subseteq \rightsquigarrow$ is harmless [26, 11, 46, 36]. We write $\overline{V}$ for the inductive extension of V to all formulas, i.e., $\overline{V}(\phi)$ is the set of all points where ϕ holds. It is well-known (and trivial to check) that the set of all formulas which hold throughout W under every valuation is a logic. Finally, when representing countermodels graphically, we will use $\bullet$ for $\rightsquigarrow$-irreflexive points and $\circ$ for $\rightsquigarrow$-reflexive ones.

2.4 Intuitionistic double negation laws

While this paper is concerned with propositional logic, in some places it will be useful to contrast double negation laws in modal and in predicate logic. We write IQC for the intuitionistic (first-order) predicate calculus. To save space and avoid distractions, the reader is referred to, e.g., Troelstra and van Dalen [52, Ch. 2] for an axiomatization.

Intuitionistic laws for $\neg\neg$ relevant from our point of view are summarized below. A fuller list can be found, e.g., in Ferreira and Oliva [21], where the reader is referred to for proofs (see also the associated Coq formalization):

▶ **Lemma 2.** *We have that:*

$$\neg\neg\phi \wedge \neg\neg\psi \dashv\vdash_{\mathsf{IPC}} \neg\neg(\neg\neg\phi \wedge \neg\neg\psi) \tag{1}$$
$$\neg\neg(\phi \wedge \psi) \dashv\vdash_{\mathsf{IPC}} \neg\neg(\neg\neg\phi \wedge \neg\neg\psi) \tag{2}$$
$$\neg\neg\phi \to \neg\neg\psi \dashv\vdash_{\mathsf{IPC}} \neg\neg(\neg\neg\phi \to \neg\neg\psi) \tag{3}$$
$$\neg\neg(\phi \vee \psi) \dashv\vdash_{\mathsf{IPC}} \neg\neg(\neg\neg\phi \vee \neg\neg\psi) \tag{4}$$
$$\neg(\neg\phi \wedge \neg\psi) \dashv\vdash_{\mathsf{IPC}} \neg\neg(\neg\neg\phi \vee \neg\neg\psi) \tag{5}$$
$$\neg\neg(\phi \to \psi) \dashv\vdash_{\mathsf{IPC}} \neg\neg(\neg\neg\phi \to \neg\neg\psi) \tag{6}$$
$$\neg(\phi \wedge \neg\psi) \dashv\vdash_{\mathsf{IPC}} \neg\neg(\phi \to \psi) \tag{7}$$
$$\neg\neg\forall x.\neg\neg\phi \dashv\vdash_{\mathsf{IQC}} \forall x.\neg\neg\phi \tag{8}$$

3 Negative translations

At least as far as modality-free languages are concerned, standard references provide numerous overviews of negative translations, e.g., [52, Ch. 2.3], [48, Ch. 6–7], [13, Ch. 2]. However, in the discussion below we are relying in particular on Ferreira and Oliva [21]. In the modal case, there is some discussion of negative translations for the basic system $\mathsf{i}_\Box\mathsf{K}$ [11, 28].

The most general conditions for a mapping $(\cdot)^t : \mathcal{L}_\Box \mapsto \mathcal{L}_\Box$ to be considered a translation function are mirroring those proposed by Gaspar [25]:

▶ **Definition 3.** Given a set of axioms $\mathcal{Z} \subseteq \mathcal{L}_\Box$ and a translation function $(\cdot)^t : \mathcal{L}_\Box \mapsto \mathcal{L}_\Box$, we say that a translation is $\mathcal{Z}$*-sane* if for any $\phi \in \mathcal{L}_\Box$, $\phi \leftrightarrow \phi^t \in \mathsf{c}_\Box\mathcal{Z}$. A $\mathcal{Z}$-sane translation is furthermore called $\mathcal{Z}$*-adequate*[5] if for any Γ and ϕ, $\vdash_{\mathsf{Nc}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi$ implies $\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma^t \Rightarrow \phi^t$.

Being both sane and adequate guarantees that $\phi \in \mathsf{c}_\Box\mathcal{Z}$ iff $\phi^t \in \mathsf{i}_\Box\mathcal{Z}$. Note that $\mathcal{Z}$-sanity is a rather trivial condition, as it transfers upwards: if $\mathsf{i}_\Box\mathcal{Z} \subseteq \mathsf{i}_\Box\mathcal{Y}$, then $\mathcal{Z}$-sanity ensures $\mathcal{Y}$-sanity; in particular, K-sanity implies $\mathcal{Z}$-sanity for any $\mathcal{Z}$. The very notion does not need to be mentioned often from now on and can be tacitly assumed: all the translations we are concerned with are K-sane. Adequacy, on the other hand, is more of a challenge; as we are going to see, when some problematic choices are made, even K-adequacy is not guaranteed.

The most direct and thorough solution is the earliest one, proposed by Kolmogorov in 1925: drop $\neg\neg$ in front of *every* subformula. This obviously extends to the modal setting:

$$\bot^{\mathsf{kol}} := \bot \qquad p^{\mathsf{kol}} := \neg\neg p \qquad (\phi \wedge \psi)^{\mathsf{kol}} := \neg\neg(\phi^{\mathsf{kol}} \wedge \psi^{\mathsf{kol}})$$
$$(\phi \vee \psi)^{\mathsf{kol}} := \neg\neg(\phi^{\mathsf{kol}} \vee \psi^{\mathsf{kol}}) \qquad (\phi \to \psi)^{\mathsf{kol}} := \neg\neg(\phi^{\mathsf{kol}} \to \psi^{\mathsf{kol}}) \qquad (\Box\phi)^{\mathsf{kol}} := \neg\neg\Box\phi^{\mathsf{kol}}$$

[5] Gaspar [25] uses the terminology "characterization" and "soundness", respectively. Similar conditions are also discussed by Ferreira and Oliva [21].

Figure 2 Frames used to show the failure of K-adequacy in Example 4 & Theorem 5 (left) and Example 6 & Theorem 7 (right), together with suitable valuations.

Of course, the Kolmogorov translation is not a model of parsimony with its liberal use of $\neg\neg$. There are two possible strategies of eliminating redundant occurrences of double negation, which we can baptize the *inner* strategy and the *outer* strategy.

The inner one is the one leading to (several variants of) the *Gödel-Gentzen* translation. then in the inductive definition of the translation, $\neg\neg$ is redundant in clauses for $\wedge$ and $\to$ owing to, respectively, Lemma 2(1) and 2(3). There are several choices one can make for $\vee$, thanks to the validity of Lemma 2(4) and 2(5); let us also note that Gödel himself relied on 2(7) to provide an alternative clause for $\to$, but few authors have followed him in this. In the predicate case, it is not necessary to prefix the universal quantifier clause with $\neg\neg$, thanks to Lemma 2(8). The perspective on $\Box$ as a restricted form of $\forall$ would seem to lead to the *naïve Gödel-Gentzen translation*:

$$\begin{array}{lll} \bot^{\mathsf{ggn}} := \bot & p^{\mathsf{ggn}} := \neg\neg p & (\phi \wedge \psi)^{\mathsf{ggn}} := \phi^{\mathsf{ggn}} \wedge \psi^{\mathsf{ggn}} \\ (\phi \vee \psi)^{\mathsf{ggn}} := \neg\neg(\phi^{\mathsf{ggn}} \vee \psi^{\mathsf{ggn}}) & (\phi \to \psi)^{\mathsf{ggn}} := \phi^{\mathsf{ggn}} \to \psi^{\mathsf{ggn}} & (\Box\phi)^{\mathsf{ggn}} := \Box\phi^{\mathsf{ggn}}. \end{array}$$

However, this translation is *not even* K*-adequate* (see also [11, p. 231–232], [28, §2.2]). A modal analogue of Lemma 2(8) fails:

▶ **Example 4.** We have that $\neg\neg\Box p \to \Box p \in \mathrm{c}_{\Box}\mathsf{K}$ but as illustrated by models based on $\mathfrak{C}^{\circ}_{\mathsf{ggn}}/\mathfrak{C}^{\bullet}_{\mathsf{ggn}}$ of Figure 2,

$$(\neg\neg\Box p \to \Box p)^{\mathsf{ggn}} = \neg\neg\Box\neg\neg p \to \Box\neg\neg p \notin \mathrm{i}_{\Box}\mathsf{K}.$$

This can be restated as:

▶ **Theorem 5.** *For any set of axioms $\mathcal{Z}$ which holds either in $\mathfrak{C}^{\circ}_{\mathsf{ggn}}$ or in $\mathfrak{C}^{\bullet}_{\mathsf{ggn}}$, the naïve Gödel-Gentzen translation* ggn *is not $\mathcal{Z}$-adequate.*

Hence, one can consider the *saturated* variant of Gödel-Gentzen (see also [11, p. 231–232], [28, Def 2.26]), which only differs in the clause for $\Box$:

$$\begin{array}{lll} \bot^{\mathsf{ggs}} := \bot & p^{\mathsf{ggs}} := \neg\neg p & (\phi \wedge \psi)^{\mathsf{ggs}} := \phi^{\mathsf{ggs}} \wedge \psi^{\mathsf{ggs}} \\ (\phi \vee \psi)^{\mathsf{ggs}} := \neg\neg(\phi^{\mathsf{ggs}} \vee \psi^{\mathsf{ggs}}) & (\phi \to \psi)^{\mathsf{ggs}} := \phi^{\mathsf{ggs}} \to \psi^{\mathsf{ggs}} & (\Box\phi)^{\mathsf{ggs}} := \neg\neg\Box\phi^{\mathsf{ggs}}. \end{array}$$

The outer strategy is the one taken by Glivenko in 1929. In the propositional case, it consists in prefixing *just the entire formula* by $\neg\neg$. In other words, one defines $\phi^{\mathsf{glv}} := \neg\neg\phi$. The double negation connective "penetrates from the outside of" an $\mathcal{L}$-formula thanks to the validity of Lemma 2(2), 2(4) and 2(6). But in this case, the strategy requires an adjustment even for IQC: $\neg\neg\forall x.\phi$ is not intuitionistically equivalent to $\neg\neg\forall x.\neg\neg\phi$. It is natural then that counterexamples can be found for $\Box$ too [11, p. 231–232]; although, as we are going to discuss in § 6, the Glivenko translation does work for a surprisingly large class of logics where $\Box$ does *not* behave like the universal quantifier and, on the other hand, there are *some* modal logics corresponding to specific universal theories where the Glivenko translation works for *some* formulas [7] (see also § 7).

▶ **Example 6.** Clearly, we have that $\Box(\neg\neg p \to p) \in \mathsf{c}_\Box\mathsf{K}$ but as illustrated by models based on $\mathfrak{C}^\circ_{\mathsf{glv}}/\mathfrak{C}^\bullet_{\mathsf{glv}}$ of Figure 2,

$$\Box(\neg\neg p \to p)^{\mathsf{glv}} = \neg\neg\Box(\neg\neg p \to p) \notin \mathsf{i}_\Box\mathsf{K}.$$

Again, this lifts to:

▶ **Theorem 7.** *For any set of axioms $\mathcal{Z}$ which holds either in $\mathfrak{C}^\circ_{\mathsf{glv}}$ or in $\mathfrak{C}^\bullet_{\mathsf{glv}}$, the Glivenko translation* glv *is not $\mathcal{Z}$-adequate.*

Thus, we need instead "saturated Glivenko": (the modal analogue of) Kuroda's 1951 translation which we can define as $\phi^{\mathsf{kur}} := \neg\neg\phi_{\mathsf{kur}}$, using an auxiliary translation $(\cdot)_{\mathsf{kur}}$ defined as $(\Box\phi)_{\mathsf{kur}} := \Box\neg\neg\phi_{\mathsf{kur}}$ and identity in other inductive clauses, i.e.,

$$\begin{array}{lll} \bot_{\mathsf{kur}} := \bot & p_{\mathsf{kur}} := p & (\phi \wedge \psi)_{\mathsf{kur}} := \phi_{\mathsf{kur}} \wedge \psi_{\mathsf{kur}} \\ (\phi \vee \psi)_{\mathsf{kur}} := \phi_{\mathsf{kur}} \vee \psi_{\mathsf{kur}} & (\phi \to \psi)_{\mathsf{kur}} := \phi_{\mathsf{kur}} \to \psi_{\mathsf{kur}} & (\Box\phi)_{\mathsf{kur}} := \Box\neg\neg\phi_{\mathsf{kur}}. \end{array}$$

3.1 Monotone modular and regular translations

Negative translations of interest to us form a subclass of *modular* ones as defined by Ferreira and Oliva [21]. We can define a somewhat narrower umbrella notion: *monotone modular translations*. Such a translation is generated by a function

$$\mathsf{cont} : ((\{\Box, \wedge, \vee, \to\} \times \{i, o\}) \cup \{\Sigma, \vdash\}) \mapsto \{0, 1\},$$

with the intuition that 0 and 1, respectively, stand for the number of occurrences of $\neg\neg$ and i and o, respectively, abbreviate *inside*/*outside*. We infix the first argument of cont as subscript. Similarly to Ferreira and Oliva [21], $\mathsf{cont}_\vdash$ stands for $\neg\neg$ used (or not) in front of the entire formula (cf. the distinction between $(\cdot)_{\mathsf{kur}}$ and $(\cdot)^{\mathsf{kur}}$ above).

Define now $\phi^t := {}^{\mathsf{cont}_\vdash \cdot \neg\neg}\phi_t$, where

$$\begin{array}{ll} \bot_t := \bot & (\phi \wedge \psi)_t := {}^{\mathsf{cont}_\wedge o \cdot \neg\neg}\big({}^{\mathsf{cont}_\wedge i \cdot \neg\neg}\phi_t \wedge {}^{\mathsf{cont}_\wedge i \cdot \neg\neg}\psi_t\big) \\ p_t := {}^{\mathsf{cont}_\Sigma \cdot \neg\neg}p & (\phi \vee \psi)_t := {}^{\mathsf{cont}_\vee o \cdot \neg\neg}\big({}^{\mathsf{cont}_\vee i \cdot \neg\neg}\phi_t \vee {}^{\mathsf{cont}_\vee i \cdot \neg\neg}\psi_t\big) \\ (\Box\phi)_t := {}^{\mathsf{cont}_\Box o \cdot \neg\neg}\Box\big({}^{\mathsf{cont}_\Box i \cdot \neg\neg}\phi_t\big) & (\phi \to \psi)_t := {}^{\mathsf{cont}_\to o \cdot \neg\neg}\big({}^{\mathsf{cont}_\to i \cdot \neg\neg}\phi_t \to {}^{\mathsf{cont}_\to i \cdot \neg\neg}\psi_t\big). \end{array}$$

▶ **Fact 8.** *Any monotone modular translation is* K*-sane, hence $\mathcal{Z}$-sane for any $\mathcal{Z} \subseteq \mathcal{L}_\Box$.*

The Glivenko translation is defined by $\mathsf{conglv}_\vdash := 1$ and 0 for all other arguments. The Kuroda translation is defined by $\mathsf{conkur}_\vdash := 1$, $\mathsf{conkur}_\Box i := 1$ and 0 elsewhere. The naïve Gödel-Gentzen translation is defined by $\mathsf{conggn}_\Sigma := 1$, $\mathsf{conggn}_\vee o := 1$ and 0 elsewhere. The saturated Gödel-Gentzen translation is defined by $\mathsf{conggs}_\Sigma := 1$, $\mathsf{conggs}_\vee o := 1$, $\mathsf{conggs}_\Box o := 1$ and 0 elsewhere. The Kolmogorov translation is defined by $\mathsf{conkol}_\Sigma := 1$, $\mathsf{conkol}_* o := 1$ for $* \in \{\Box, \wedge, \vee, \to\}$ and 0 elsewhere. There is a natural *weak saturation ordering* on such translations: $t \le t'$ whenever cont is pointwise below cont' (obviously, $0 \le 1$). Thus, $\mathsf{ggn} \le \mathsf{ggs} \le \mathsf{kol}$ and $\mathsf{glv} \le \mathsf{kur}$. Any monotone modular translation weakly saturating either ggs or kur is called *regular*. As the question of adequacy of regular translations will be of central importance below, let us introduce a name for it.

▶ **Definition 9.** A logic $\mathsf{i}_\Box\mathcal{Z}$ is *$\neg\neg$-complete* if some/any regular t is adequate for it.

▶ **Theorem 10.** *Any regular translation t is* K*-adequate, i.e., $\mathsf{i}_\Box\mathsf{K}$ is $\neg\neg$-complete. Furthermore, for every $\phi \in \mathcal{L}_\Box$ we have that $\vdash_{\mathsf{K}} \phi^t \leftrightarrow \neg\neg\phi^t$ and for any other regular translation t', we have that $\vdash_{\mathsf{K}} \phi^t \leftrightarrow \phi^{t'}$.*

As said above, $\neg\neg$-completeness of the minimal system $\mathsf{i}_\Box\mathsf{K}$ has already been noted in the literature [11, 28]. But not all logics are $\neg\neg$-complete.

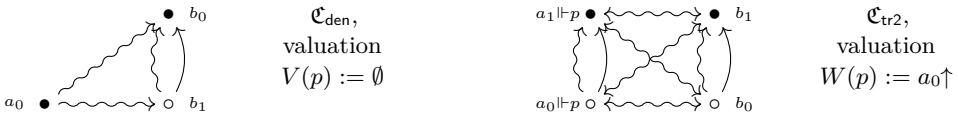


Figure 3 Frames used to show the failure of $\neg\neg$-completeness in Example 11 & Theorem 12 (left) and Example 13 & Theorem 14 (right), together with suitable valuations.

4 Failure of $\neg\neg$-completeness

This section presents counterexamples illustrating that there are logics for which regular translations are not adequate. Apart from counterexamples in the previous section, this is the only place where we need to use Kripke semantics.

▶ **Example 11.** Consider the frame $\mathfrak{C}_{\mathsf{den}}$ with valuation V depicted in Figure 3. We have that $\mathfrak{C}_{\mathsf{den}} \Vdash \mathsf{i}_\Box\mathsf{C4}$, where $\mathsf{C4} := \Box\Box p \to \Box p$. But

$$\mathfrak{C}_{\mathsf{den}}, V, a_0 \nVdash \neg\neg\Box\neg\neg\Box\neg\neg p \to \neg\neg\Box\neg\neg p.$$

To see this, observe that $\overline{V}(\neg\neg p) = \emptyset$, $\overline{V}(\Box\neg\neg p) = \{b_0\} = b_0{\uparrow}$ and $\overline{V}(\neg\neg\Box\neg\neg p) = \{b_0, b_1\} = b_1{\uparrow}$, whereas $\overline{V}(\Box\neg\neg\Box\neg\neg p)$ is the entire carrier of $\mathfrak{C}_{\mathsf{den}}$ and hence so is $\overline{V}(\neg\neg\Box\neg\neg\Box\neg\neg p)$.

This lifts to a theorem about an interval of logics:

▶ **Theorem 12.** *No extension of* $\mathsf{i}_\Box\mathsf{K}$ *contained between* $\mathsf{i}_\Box\mathsf{C4}$ *and the logic of* $\mathfrak{C}_{\mathsf{den}}$ *is* $\neg\neg$*-complete.*

Is this an isolated example? As it turns out, $\neg\neg$-completeness can fail for other rather simple *modal reduction principles* [58, §4.5] as well.

▶ **Example 13.** Consider $\flat\mathsf{4} := \Box\Box p \to \Box\Box\Box p$ and the frame $\mathfrak{C}_{\mathsf{tr2}}$ with valuation W depicted in Figure 3. We can easily show that $\mathfrak{C}_{\mathsf{tr2}} \Vdash \flat\mathsf{4}$. On the other hand,

$$\mathfrak{C}_{\mathsf{tr2}}, W, a_0 \nVdash \neg\neg\Box\neg\neg\Box\neg\neg p \to \neg\neg\Box\neg\neg\Box\neg\neg\Box\neg\neg p.$$

This is verified as follows:

$$\begin{array}{ll} a_1{\uparrow} = \overline{W}(\Box\neg\neg\Box\neg\neg p), & b_1{\uparrow} = \overline{W}(\Box\neg\neg p) = \overline{W}(\Box\neg\neg\Box\neg\neg\Box\neg\neg p), \\ a_0{\uparrow} = W(p) = \overline{W}(\neg\neg p) = \overline{W}(\neg\neg\Box\neg\neg\Box\neg\neg p), & b_0{\uparrow} = \overline{W}(\neg\neg\Box\neg\neg p) = \overline{W}(\neg\neg\Box\neg\neg\Box\neg\neg\Box\neg\neg p). \end{array}$$

▶ **Theorem 14.** *No logic contained between* $\mathsf{i}_\Box\flat\mathsf{4}$ *and the logic of* $\mathfrak{C}_{\mathsf{tr2}}$ *is* $\neg\neg$*-complete.*

4.1 Aside: with great power comes great inadequacy

While in this paper we focus almost entirely on the simplest possible case of $\Box$-logics, inadequacy can be a more dramatic phenomenon for very natural systems with more connectives, e.g., residuated binary ones of BI: the logic of *bunched implications* [42, 45]. On the one hand, Galmiche et al. [24] have shown that BI is decidable; this result allows strengthening and generalizations [22]. On the other hand, the undecidability of its classical extension BBI follows already from results of Kurucz et al. [33, 2] (shown using von Neumann's *n-frames* originating in projective geometry) and has been recently rediscovered and extended to logics determined by concrete heap models using more computational techniques [12, 35].

Hence, no *recursive* translation from BBI to BI (much less a modular negative one) can be adequate; otherwise, one could use such a translation to define a decision procedure for BBI.

5 Syntactic criteria of $\neg\neg$-completeness

Given that $\neg\neg$-completeness can fail so dramatically for very natural axioms, it is natural to ask how one can obtain general *positive* results. First, let us analyze where the problem comes from. By definition, a logic fails to be $\neg\neg$-complete iff there exists $\phi \in \mathrm{c}_\Box\mathcal{Z}$ s.t. $\phi^t \notin \mathrm{i}_\Box\mathcal{Z}$. It would seem that nothing forces this ϕ to be itself an axiom, i.e., a member of $\mathcal{Z}$, but this is precisely what we see in the counterexamples in the preceding section. The following theorem shows it is not a coincidence – and provides us with an useful criterion of $\neg\neg$-completeness.

▶ **Theorem 15** (Regular Adequacy)**.**

- *A logic* $\mathrm{i}_\Box\{\zeta\}$ *is* $\neg\neg$*-complete iff* $\zeta^t \in \mathrm{i}_\Box\{\zeta\}$ *for some/any regular translation* t*.*
- *A logic* $\mathrm{i}_\Box\mathcal{Z}$ *is* $\neg\neg$*-complete whenever for some/any regular translation* t *and for any* $\zeta \in \mathcal{Z}$*, it holds that* $\zeta^t \in \mathrm{i}_\Box\mathcal{Z}$*.*

Proof. Fix $t = \mathsf{ggs}$. We need to show that whenever $\vdash_{\mathsf{Nc}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi$, $\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma^{\mathsf{ggs}} \Rightarrow \phi^{\mathsf{ggs}}$. The proof proceeds by induction on the derivation of $\vdash_{\mathsf{Nc}_\Box\mathcal{Z}} \Gamma \Rightarrow \phi$. The assumption of the theorem guarantees the AxSb case. The cases of IPC rules are straightforward and known, with the case of $(\vee_E)$ requiring, as usual, somewhat more bookkeeping. Let us show the modal case, i.e., $\Box_\mathsf{K}$. Our goal is to derive $\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma^{\mathsf{ggs}} \Rightarrow \neg\neg\Box\psi^{\mathsf{ggs}}$ under the assumption that

$$\vdash_{\mathsf{Nc}_\Box\mathcal{Z}} \Gamma \Rightarrow \Box\phi_1, \ldots, \vdash_{\mathsf{Nc}_\Box\mathcal{Z}} \Gamma \Rightarrow \Box\phi_n \text{ and } \vdash_{\mathsf{Nc}_\Box\mathcal{Z}} \phi_1, \ldots, \phi_n \Rightarrow \psi$$

and hence, by IH,

$$\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma^{\mathsf{ggs}} \Rightarrow \neg\neg\Box\phi_1^{\mathsf{ggs}}, \ldots, \vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma^{\mathsf{ggs}} \Rightarrow \neg\neg\Box\phi_n^{\mathsf{ggs}} \text{ and } \vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \phi_1^{\mathsf{ggs}}, \ldots, \phi_n^{\mathsf{ggs}} \Rightarrow \psi^{\mathsf{ggs}}.$$

But now it is enough to observe that the following rule is derivable:

$$\neg\neg\Box_\mathsf{K}\ \frac{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \neg\neg\Box\phi_1 \qquad \ldots \vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \neg\neg\Box\phi_n \qquad \vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \phi_1, \ldots, \phi_n \Rightarrow \psi}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Gamma \Rightarrow \neg\neg\Box\psi}\ .$$ ◀

Thus, whenever a given finite set of axioms $\mathcal{Z}$ axiomatizes a logic with a known decision algorithm, one can use the decision procedure for $\mathrm{i}_\Box\mathcal{Z}$ to check its $\neg\neg$-completeness: i.e., by checking if the Kolmogorov translation (or any other regular translation, e.g., Kuroda for simplicity) of each axiom from $\mathcal{Z}$ is a member of $\mathrm{i}_\Box\mathcal{Z}$. Of course, one needs to keep in mind that the problem whether a given formula axiomatizes a decidable logic is undecidable itself (see, e.g., [13, Ch. 17], [58, §3] for references). Nevertheless, as we are going to discuss now, Theorem 15 does yield general positive results on $\neg\neg$-completeness of logics with axioms of a specific syntactic shape.

▶ **Definition 16.** We call $\beta \in \mathcal{L}_\Box$ a ($\neg\neg$-) *pre-envelope* (*in* $\mathrm{i}_\Box\mathcal{Z}$) if $\neg\neg s_{\neg\neg}(\beta) \vdash_{\mathrm{i}_\Box\mathcal{Z}} \beta^{\mathsf{kol}}$; recall that $s_{\neg\neg}(\beta)$ is a substitution replacing all p occurring in β by their double negations. When not stated specifically, we will take $\mathrm{i}_\Box\mathcal{Z}$ to be $\mathrm{i}_\Box\mathsf{K}$. Analogously, we call $\beta \in \mathcal{L}_\Box$ a ($\neg\neg$-) *post-envelope* (*in* $\mathrm{i}_\Box\mathcal{Z}$) if $\beta^{\mathsf{kol}} \vdash_{\mathrm{i}_\Box\mathcal{Z}} \neg\neg s_{\neg\neg}(\beta)$. A $\neg\neg$-*envelope* is a formula which is both pre- and post-envelope, i.e., a formula $\beta \in \mathcal{L}_\Box$ s.t. $\neg\neg s_{\neg\neg}(\beta) \dashv\vdash_{\mathrm{i}_\Box\mathcal{Z}} \beta^{\mathsf{kol}}$. An *enveloped implication* is of the form $\beta \to \gamma$ where β is a post-envelope and γ is a pre-envelope.

Clearly, a $\neg\neg$-envelope can be consider a special (or degenerate) case of an enveloped implication. To illustrate these notions, recall that a *shallow formula* is one with no nesting of $\Box$, i.e., one where every $p \in \Sigma$ is within the scope of at most one $\Box$, whereas a *box-free formula* is one without any occurrences of $\Box$ at all. Now we have:

▶ **Lemma 17** (Envelope Criteria).

- *A box-free formula is a $\dashv\neg$-envelope*
- *A shallow formula with no disjunction under box is a $\dashv\neg$-envelope*
- *An implication-free formula is a pre-envelope*
- *A negation of a pre-envelope is a post-envelope*

Proof. An exercise in induction. See the associated formalization for details. ◀

▶ **Theorem 18** (Enveloped Implications). *Any logic axiomatized by enveloped implications is $\dashv\neg$-complete.*

Proof. By Theorem 15, it is enough to show that $\zeta^{\mathsf{kol}} \in \mathrm{i}_{\Box}\mathcal{Z}$ for any $\zeta \in \mathcal{Z}$, where $\zeta = \beta \to \gamma$, β is a post-envelope and γ is a pre-envelope:

$$\dfrac{\dfrac{\dfrac{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \dashv\neg s_{\dashv\neg}(\beta) \to \dashv\neg s_{\dashv\neg}(\gamma), \beta^{\mathsf{kol}} \Rightarrow \gamma^{\mathsf{kol}}}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Rightarrow (\dashv\neg s_{\dashv\neg}(\beta) \to \dashv\neg s_{\dashv\neg}(\gamma)) \to \beta^{\mathsf{kol}} \to \gamma^{\mathsf{kol}}}\,{\to_I}}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Rightarrow \dashv\neg(\dashv\neg s_{\dashv\neg}(\beta) \to \dashv\neg s_{\dashv\neg}(\gamma)) \to \dashv\neg(\beta^{\mathsf{kol}} \to \gamma^{\mathsf{kol}})}\,{\dashv\neg\to_1} \qquad \dfrac{\dfrac{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Rightarrow \beta \to \gamma}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Rightarrow s_{\dashv\neg}(\beta) \to s_{\dashv\neg}(\gamma)}\,\textsc{AxSb}}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Rightarrow \dashv\neg(\dashv\neg s_{\dashv\neg}(\beta) \to \dashv\neg s_{\dashv\neg}(\gamma))}\,{\dashv\neg\to_2}}{\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \Rightarrow \dashv\neg(\beta^{\mathsf{kol}} \to \gamma^{\mathsf{kol}})}\,{\to_E}.$$

Thus, we just need to show that $\vdash_{\mathsf{Ni}_\Box\mathcal{Z}} \dashv\neg s_{\dashv\neg}(\beta) \to \dashv\neg s_{\dashv\neg}(\gamma), \beta^{\mathsf{kol}} \Rightarrow \gamma^{\mathsf{kol}}$. But this follows using the definitions of pre- and post-envelopes. ◀

Theorem 18 jointly with Lemma 17 yields $\dashv\neg$-completeness of numerous logics, including, for example, many of those given in Table 2 of Litak [36] or in Theorem 10 of Sotirov [49]. Here are just some examples, with somewhat random names (although inasmuch as possible we respect those already used in existing literature):

▶ **Corollary 19.** *$\dashv\neg$-completeness holds for any logic axiomatized over $\mathrm{i}_{\Box}\mathsf{K}$ by any combination of the following formulas:*

R	$p \to \Box p$,	CB	$\Box p \to (q \to p) \vee q$,	$\flat$em	$\Box p \vee \Box\neg p$,
4	$\Box p \to \Box\Box p$,	NV	$\neg\Box\bot$,	em$\flat$	$\Box p \vee \neg\Box p$,
T	$\Box p \to p$,	NNV	$\dashv\neg\Box\bot$,	T$^{\dashv\neg}$	$\Box p \to \dashv\neg p$,
coK	$(\Box p \to \Box q) \to \Box(p \to q)$,	$\flat$R	$p \to \Box\Box p$,	T$^{\neg}$	$\Box\neg p \to \neg p$,
$\flat$Lin	$\Box(p \to q) \vee \Box(q \to p)$,	Lin$\flat$	$(\Box p \to q) \vee (\Box q \to p)$,	wem$\flat^{\neg}$	$\neg\Box p \vee \neg\Box\neg p$,

or any superintuitionistic axiom, i.e., a formula in modality-free $\mathcal{L}$.

As an example of a standard logic obtained by a combination of axioms given in Corollary 19, consider $\mathrm{i}_{\Box}\mathsf{S4} = \mathrm{i}_{\Box}\mathsf{4} + \mathsf{T}$.

Counterexamples in § 4 were easily seen to produce entire intervals of $\dashv\neg$-incomplete logics (Theorems 12 and 14). In contrast, Corollary 19 does not automatically lead to non-trivial examples of entire intervals of $\dashv\neg$-*complete* logics, especially not to logics all of whose extensions are $\dashv\neg$-complete. Corollary 19 mentions, e.g., $\mathrm{i}_{\Box}\mathsf{4}$ and Example 11 together with Theorem 12 provide an example of an extension of $\mathrm{i}_{\Box}\mathsf{4}$ for which $\dashv\neg$-completeness fails: the logic of $\mathfrak{C}_{\mathsf{den}}$. However, Corollary 19 also mentions $\mathrm{i}_{\Box}\mathsf{R}$ – and for this system, we can do much better. As it turns out, this is related to another subject: for some classes of logics, irregular translations can be adequate.

6 Strength makes irregularity adequate

The failure of the Glivenko translation amounts to the failure of the "outer strategy" described in § 3 to penetrate through $\Box$. The axiom $\forall x.\neg\neg\phi \to \neg\neg\forall x.\phi$ which ensures this strategy succeeds in the predicate case is thus called the *Double Negation Shift* [52, Ch. 2] and in some references also *Kuroda principle* (*scheme, axiom*) – see, e.g., [20] for an example of both names in use. It is natural to use similar name(s) for its modal analogue. In other words, the (*modal*) *Double Negation Shift* (or the *modal Kuroda axiom*) is

DNS: $\quad \Box\neg\neg p \to \neg\neg\Box p.$

The corresponding logic is naturally denoted as $\mathsf{i}_\Box\mathsf{DNS}$. Clearly, we have that for any $\mathsf{i}_\Box\mathcal{Z}$ extending $\mathsf{i}_\Box\mathsf{DNS}$ and any $\phi \in \mathcal{L}_\Box$, $\neg\neg\Box\neg\neg\phi \leftrightarrow \neg\neg\Box\phi \in \mathsf{i}_\Box\mathcal{Z}$. In fact, it is straightforward to see that these two axiom schemes are equivalent:

▶ **Lemma 20.** DNS *and* $\neg\neg\Box\neg\neg p \leftrightarrow \neg\neg\Box p$ *axiomatize the same logic.*

We have an analogue of Exercise 2.3.3 in Troelstra and van Dalen [52]:

▶ **Theorem 21.** *In any extension of* $\mathsf{i}_\Box\mathsf{DNS}$*, the Glivenko translation becomes equivalent to the regular ones, i.e., for any regular* t *and every* ϕ*, we have that:* $\vdash_{\mathsf{i}_\Box\mathsf{DNS}} \phi^t \leftrightarrow \phi^{\mathsf{glv}}$.

Hence, we have $\neg\neg$-completeness of *all logics* in which the Kuroda axiom is derivable:

▶ **Theorem 22.** *Any* t *saturating* glv *is adequate for any extension of* $\mathsf{i}_\Box\mathsf{DNS}$.

Proof Sketch. It is straightforward to extend Theorem 21 to any translation saturating Glivenko, i.e., for any extension of $\mathsf{i}_\Box\mathsf{DNS}$ each such translation is equivalent to, e.g., the Kolmogorov translation. Hence, it is enough to show that kol is adequate. Now apply Theorem 15 and consider an arbitrary formula $\zeta \in \mathsf{i}_\Box\mathcal{Z}$, where $\mathcal{Z}$ is a freely chosen axiomatization of the logic in question. We need to show that ζ^{kol} belongs to $\mathsf{i}_\Box\mathcal{Z}$. But Theorem 21 says that $\zeta^{\mathsf{kol}} \dashv\vdash_{\mathsf{i}_\Box\mathsf{DNS}} \neg\neg\zeta$ and $\zeta \to \neg\neg\zeta$ holds in any extension of $\mathsf{i}_\Box\mathsf{K}$. ◀

Are extensions of $\mathsf{i}_\Box\mathsf{DNS}$ of interest? Consider $\mathsf{i}_\Box\mathsf{R} = \mathsf{i}_\Box\mathsf{K} + p \to \Box p$, which can be seen as the logic of *applicative functors*, also known as *idioms* [37]. Our presently used name R follows the usage in Litak [36], which in turn uses the same name as Fairtlough and Mendler [19]; another possible justification for this name would come from the "return" law of monads. An alternative name would be S, coming from, on the one hand, the *strong Löb axiom*

SL: $(\Box p \to p) \to p$,

and on the other hand from categorical "strength" of $\Box$ thought of as a functor, whose trace on the level of type/object inhabitation is

Str: $p \wedge \Box q \to \Box(p \wedge q)$.

It is easy to see that R and Str are interderivable over $\mathsf{i}_\Box\mathsf{K}$.[6] Regarding the strong Löb axiom, the derivation of S in $\mathsf{i}_\Box\mathsf{SL}$ can be found, e.g., in Milius and Litak [38, §3] (in a categorical disguise). It is a simplified variant of the deduction of 4 from the ordinary Löb axiom found independently by de Jongh, Kripke and Sambin in mid-1970's, cf. [9, Th 18]. SL is an axiom whose importance has been first noticed in provability logics HA^* and PA^* [54, 29]. Its recent popularity deriving from Nakano [40, 41] comes from its rôle in guarding (co-)recursion and

[6] We decided against the use of S to avoid confusion with standard modal names S4 and S5, which come from unfortunate historical coincidences.

(co-)induction.[7] Another important class of extensions of $\mathsf{i}_\Box\mathsf{R}$ is provided by extensions of $\mathsf{PLL} := \mathsf{i}_\Box\mathsf{R} + \mathsf{C4}$ (Propositional Lax Logic [19]): the logic of the *Grothendieck topology* [26], but also *access control* and the Curry-Howard counterpart of Moggi's *computational metalanguage* (see [36] for references). Recall that above $\mathsf{i}_\Box\mathsf{K}$, C4 was one of our flagship examples of an axiom failing $\neg\neg$-completeness. Finally, Artemov and Protopopescu [4] use the BHK interpretation to justify the epistemic importance of certain extension of $\mathsf{i}_\Box\mathsf{R}$ and PLL. In short, despite the non-classical character of these modalities, we are dealing with a large class of important extensions of $\mathsf{i}_\Box\mathsf{K}$; as mentioned in the introduction, this puts in doubt Simpson's Requirement 3 [46, Ch 3.2]. And we have:

▶ **Theorem 23.** DNS *is a theorem of* $\mathsf{i}_\Box\mathsf{R}$.

Proof. First, observe that

$$\dfrac{\dfrac{\vdash_{\mathsf{Ni}_\Box\mathsf{R}} \Box\neg\neg\phi, \neg\Box\phi \Rightarrow \Box\phi \qquad \vdash_{\mathsf{Ni}_\Box\mathsf{R}} \Box\neg\neg\phi, \neg\Box\phi \Rightarrow \neg\Box\phi}{\vdash_{\mathsf{Ni}_\Box\mathsf{R}} \Box\neg\neg\phi, \neg\Box\phi \Rightarrow \bot}\,{\to_E}}{\vdash_{\mathsf{Ni}_\Box\mathsf{R}} \Box\neg\neg\phi \Rightarrow \neg\Box\phi.}\,{\to_I}$$

Hence, we just need to derive $\vdash_{\mathsf{Ni}_\Box\mathsf{R}} \Box\neg\neg\phi, \neg\Box\phi \Rightarrow \Box\phi$. For this, it is enough to show

$$\vdash_{\mathsf{Ni}_\Box\mathsf{R}} \Box\neg\neg\phi, \neg\Box\phi \Rightarrow \Box\bot.$$

This in turn is obtained via

$$\dfrac{\vdash_{\mathsf{Ni}_\Box\mathsf{R}} \Box\neg\neg\phi, \neg\Box\phi \Rightarrow \Box\neg\phi \qquad \vdash_{\mathsf{Ni}_\Box\mathsf{R}} \Box\neg\neg\phi, \neg\Box\phi \Rightarrow \Box\neg\neg\phi \qquad \vdash_{\mathsf{Ni}_\Box\mathsf{R}} \neg\neg\phi, \neg\phi \Rightarrow \bot}{\vdash_{\mathsf{Ni}_\Box\mathsf{R}} \Box\neg\neg\phi, \neg\Box\phi \Rightarrow \Box\bot.}\,{\Box_\mathsf{K}}$$

Only the first premise is interesting, and this is the only part of the proof where the R axiom is actually used:

$$\dfrac{\vdash_{\mathsf{Ni}_\Box\mathsf{R}} \neg\Box\phi \Rightarrow \neg\phi \qquad \dfrac{}{\vdash_{\mathsf{Ni}_\Box\mathsf{R}} \neg\Box\phi \Rightarrow \neg\phi \to \Box\neg\phi}\,\textsc{AxSb}}{\vdash_{\mathsf{Ni}_\Box\mathsf{R}} \neg\Box\phi \Rightarrow \Box\neg\phi.}\,{\to_E}$$

The derivation of $\vdash_{\mathsf{Ni}_\Box\mathsf{R}} \neg\Box\phi \Rightarrow \neg\phi$ can be now left as an exercise. ◀

The above deduction should look familiar to readers acquainted with CPS reasoning and the use of control operators.

▶ **Corollary 24.** *Any* t *saturating* glv *is adequate for any extension of* $\mathsf{i}_\Box\mathsf{R}$*; in particular, each such logic is* $\neg\neg$*-complete.*

▶ Remark 25. Extensions of $\mathsf{i}_\Box\mathsf{R}$ have a rather trivial "double negation core": $\mathsf{c}_\Box\mathsf{K}$ extended with the "strength" or "return" axiom R is equipollent with CPC enriched with an additional propositional constant (corresponding to $\Box\bot$). There are only three consistent (and coNP-complete) logics of this kind, whereas the lattice of extensions of $\mathsf{i}_\Box\mathsf{R}$ is uncountable and

[7] It is used to ensure productivity in (co-)programming and on the metalevel – in semantic reasoning about programs involving higher-order store or a combination of impredicative quantification with recursive types. There are too many recent references to quote here, so let us just point out that Milius and Litak [38] provide a general framework for various categorical models in the literature, ranging from ultrametric spaces to the *topos of trees* of Birkedal and coauthors [8]. For more references, see also [36].

richer than the lattice of extensions of IPC, which can be identified with $\mathsf{PLL} + \Box p \leftrightarrow p$. In particular, logics extending $\mathsf{i}_\Box\mathsf{R}$ can happen to be undecidable. Hence, in many ways the gap between $\mathsf{i}_\Box\mathsf{R}$ and $\mathsf{c}_\Box\mathsf{R}$ is more dramatic than that between $\mathsf{i}_\Box\mathsf{S4}$ and $\mathsf{c}_\Box\mathsf{S4}$. One can see the fact that the "double negation core" of any extension of $\mathsf{i}_\Box\mathsf{R}$ collapses so much information as a deeper reason behind Corollary 24.

7 Conclusions and future work

While we consciously restricted our attention to syntactic criteria and syntactic proofs in this paper, it is also possible to prove positive general results for adequacy and $\neg\neg$-completeness in terms of stability under $\uparrow$-*cofinal* and $\uparrow$-*generated subframes*; indeed, readers familiar with modal logic probably noticed that counterexamples in § 4 fail to be *subframe logics* (cf. e.g, [59, §1.8]). In future work, we will discuss whether this approach leads to a more general characterization of adequacy than Theorem 18.

On another note, one of the reasons why modular translations of Ferreira and Oliva [21] are defined in a more general way than our monotone modular translations in § 3.1 is that they cover also the so-called Krivine(-Streicher-Reus) translation. This translation dualizes connectives, in particular uses $\forall$ and $\exists$ to translate each other. It would seem natural to study such translations, but this appears sensible only in the presence of $\Diamond$ suitably related to $\Box$.[8] In this connection, let us recall that Bezhanishvili [7] studies the scope of Glivenko-type theorems for specific formulas in extensions of Prior's MIPC, i.e., logics equipped with $\Box$ and $\Diamond$ which behave as close as possible to genuine $\forall$ and $\exists$ for concrete theories in IQC. Furthermore, it would be of obvious interest to study still more complicated supplies of connectives and axioms than those involving $\Box$ and $\Diamond$; nevertheless, as discussed in § 4.1 using BI as an example, things can go very wrong for very natural logics. The scope of any general positive results will be by nature limited, and a suitable proof-theoretic setup and associated proof-assistant formalization would be more complicated. Speaking of substructural connectives, however, it would seem natural to merge the present line of research with syntactic research on substructural negative translations by Ono and coauthors [43, 20].

Still a different direction would be to pick up the theme already signalled in Footnote 2: $\neg\neg$ can and has been studied as a modality in its own right [10, 17], especially over negation-free and bottom-free intutionistic syntax. Algebraically, one can see $\neg\neg$ as a *nucleus* and categorically – as (inhabitation/object level trace of) the *continuation monad*. The corresponding generalization of our study of modal negative translations would connect with the work of Aczel [1] and Escardó and Oliva [18]. Replacement of the intuitionistic logic with the minimal logic would be of interest given the significance of the latter in terms of control operators [3]. Surely enough, one cannot expect each and every possible modal axiom to enjoy some computational significance. Pfenning and Davies [44] stress the importance of adequate introduction and elimination rules in this context, Martin-Löf style; contrast this with our discussion in § 2.2 of the unavailability of an apparatus producing such rules for arbitrary axioms without any syntactic restrictions on their shape. At the very least, however, we hope that the present development provides good limitative results and a general

[8] An interesting intermediate challenge was posed by one of the referees: broadening the framework somewhat to allow modal versions of the double negation translations mKu [21, §1], E [21, §5] or N_1 [25], which contrarily to Krivine translation can be still introduced with a single modality. Such a generalization would require more technicalities; e.g. for E, the straightforward ordering proposed in § 3.1 and some of the statements and proofs will need to become more complicated.

framework for investigating the relationship between constructive and classical variants of concrete axioms/inhabitation laws. More generally, we believe that our study illustrates the availability of purely constructive and syntactic methods in modal logic understood as "die Klassentheorie" [58]. We do not claim that such methods can or should *replace* algebraic, topological and model-theoretic ones. Nevertheless, it is good to remember that they can be employed in the service of what Wolter and Zakharyaschev [58] call the *global view* and *intuitionistic* modal logics seem a particularly natural target.

Acknowledgements. We would like to thank the reviewers, Wesley H. Holliday, and Lutz Schröder for their helpful and constructive comments, which helped us to improve the presentation in the final version. We also thank Erwin R. Catesbeiana for his non-constructive comments about constructive negation.

References

1 Peter Aczel. The Russell-Prawitz modality. *MSCS*, 11(4):541–554, 2001.

2 H. Andréka, Á. Kurucz, I. Németi, I. Sain, and A. Simon. Causes and remedies for undecidability in arrow logics and in multi-modal logics. In Maarten Marx, László Pólos, and Michael Masuch, editors, *Arrow Logic and Multi-Modal Logic*, Stud. Logic Lang. Inform., pages 63–100. CSLI Publications, 1996.

3 Zena M. Ariola and Hugo Herbelin. Minimal classical logic and control operators. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Proceedings of ICALP*, volume 2719 of *LNCS*, pages 871–885. Springer, 2003. doi:10.1007/3-540-45061-0_68.

4 Sergei Artemov and Tudor Protopopescu. Intuitionistic epistemic logic. *Rev. Symb. Logic*, 9(2):266–298, 2016. doi:10.1017/S1755020315000374.

5 Gianluigi Bellin. A system of natural deduction for GL. *Theoria*, 51(2):89–114, 1985. doi:10.1111/j.1755-2567.1985.tb00089.x.

6 Gianluigi Bellin, Valeria de Paiva, and Eike Ritter. Extended Curry-Howard correspondence for a basic constructive modal logic. In *Proceedings of Methods for Modalities*, 2001. URL: http://profs.sci.univr.it/~bellin/m4m.ps.

7 Guram Bezhanishvili. Glivenko type theorems for intuitionistic modal logics. *Stud. Logica*, 67(1):89–109, 2001. doi:10.1023/A:1010577628486.

8 Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. First steps in synthetic guarded domain theory: Step-indexing in the topos of trees. *LMCS*, 8:1–45, 2012.

9 George Boolos. *The Logic of Provability*. Cambridge University Press, 1993.

10 Milan Božić and Kosta Došen. Axiomatizations of intuitionistic double negation. *B. Sect. Logic*, 12(2):99–102, 1983.

11 Milan Božić and Kosta Došen. Models for normal intuitionistic modal logics. *Stud. Logica*, 43(3):217–245, 1984. URL: http://www.jstor.org/stable/20015164.

12 James Brotherston and Max Kanovich. Undecidability of propositional separation logic and its neighbours. *J. ACM*, 61(2):14:1–14:43, 04 2014. doi:10.1145/2542667.

13 Alexander Chagrov and Michael Zakharyaschev. *Modal Logic*. Number 35 in Oxford Logic Guides. Clarendon Press, 1997.

14 Agata Ciabattoni, Nikolaos Galatos, and Kazushige Terui. From axioms to analytic rules in nonclassical logics. In *Proceedings of LiCS*, pages 229–240. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.39.

15 Agata Ciabattoni, Lutz Straßburger, and Kazushige Terui. Expanding the realm of systematic proof theory. In *Proceedings of CSL*, pages 163–178, Berlin, Heidelberg, 2009. Springer-Verlag.

16 Valeria de Paiva and Eike Ritter. Basic constructive modality. In Jean-Yves Beziau and Marcelo Esteban Coniglio, editors, *Logic Without Frontiers- Festschrift for Walter Alexandre Carnielli on the occasion of his 60th birthday*, pages 411–428. College Publications, 2011.

17 Kosta Došen. Modal translations and intuitionistic double negation. *Logique et Analyse*, 29(113):81–94, 1986. URL: http://www.jstor.org/stable/44084152.

18 Martín Escardó and Paulo Oliva. The Peirce translation and the Double Negation Shift. In Fernando Ferreira, Benedikt Löwe, Elvira Mayordomo, and Luís Mendes Gomes, editors, *Proceedings of CiE*, pages 151–161. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-13962-8_17.

19 Matt Fairtlough and Michael Mendler. Propositional lax logic. *Inform. and Comput.*, 137(1):1–33, 1997.

20 Hadi Farahani and Hiroakira Ono. Glivenko theorems and negative translations in substructural predicate logics. *Arch. Math. Log.*, 51(7-8):695–707, 2012. doi:10.1007/s00153-012-0293-8.

21 Gilda Ferreira and Paulo Oliva. On the relation between various negative translations. In Ulrich Berger and Helmut Schwichtenberg, editors, *Logic, Construction, Computation*, volume 3 of *Mathematical Logic Series*, pages 227–258. Ontos-Verlag, 2012.

22 Nikolaos Galatos and Peter Jipsen. Distributive residuated frames and generalized bunched implication algebras. *Algebr. Univ.*, to appear.

23 Nikolaos Galatos and Hiroakira Ono. Glivenko theorems for substructural logics over FL. *J. Symb. Logic*, 71(4):1353–1384, 2006. URL: http://www.jstor.org/stable/27588518.

24 Didier Galmiche, Daniel Méry, and David J. Pym. The semantics of BI and resource tableaux. *MSCS*, 15(6):1033–1088, 2005. doi:10.1017/S0960129505004858.

25 Jaime Gaspar. Negative translations not intuitionistically equivalent to the usual ones. *Stud. Logica*, 101(1):45–63, 2013. doi:10.1007/s11225-011-9367-6.

26 Robert I. Goldblatt. Grothendieck topology as geometric modality. *MLQ*, 27(31–35):495–529, 1981. doi:10.1002/malq.19810273104.

27 Timothy G. Griffin. A formulae-as-type notion of control. In *Proceedings of POPL*, Proceedings of POPL, pages 47–58, New York, NY, USA, 1990. ACM. doi:10.1145/96709.96714.

28 Wesley Halcrow Holliday. Possibility frames and forcing for modal logic (June 2016), 2016. submitted eScholarship manuscript: http://www.escholarship.org/uc/item/9v11r0dq.

29 Rosalie Iemhoff. *Provability logic and admissible rules*. phdthesis, University of Amsterdam, 2001.

30 Yoshihiko Kakutani. Call-by-name and call-by-value in normal modal logic. In Zhong Shao, editor, *Proceedings of APLAS*, volume 4807 of *LNCS*, pages 399–414. Springer, 2007. doi:10.1007/978-3-540-76637-7_27.

31 Daisuke Kimura and Yoshihiko Kakutani. Classical natural deduction for S4 modal logic. *New Generation Comput.*, 29(1):61–86, 2011. doi:10.1007/s00354-010-0099-3.

32 Clemens Kupke, Alexander Kurz, and Yde Venema. Stone coalgebras. *Theoret. Comput. Sci.*, 327(1):109–134, 2004. doi:10.1016/j.tcs.2004.07.023.

33 Á. Kurucz, I. Németi, I. Sain, and A. Simon. Decidable and undecidable logics with a binary modality. *JoLLI*, 4(3):191–206, 1995. URL: http://www.jstor.org/stable/40180071.

34 Charles H. Lambros. A shortened proof of Sobociński's theorem concerning a restricted rule of substitution in the field of propositional calculi. *Notre Dame J. Form. L.*, 20(1):112–114, 01 1979. doi:10.1305/ndjfl/1093882409.

35 Dominique Larchey-Wendling and Didier Galmiche. Nondeterministic phase semantics and the undecidability of boolean BI. *ACM Trans. Comput. Logic*, 14(1):6:1–6:41, 02 2013. doi:10.1145/2422085.2422091.

36 Tadeusz Litak. Constructive modalities with provability smack. In Guram Bezhanishvili, editor, *Leo Esakia on duality in modal and intuitionistic logics*, volume 4 of *Outstanding Contributions to Logic*, pages 179–208. Springer, 2014. *Author's Cut*: https://www8.cs.fau.de/ext/litak/esakiaarxivfull.pdf. doi:10.1007/978-94-017-8860-1_8.

37 Conor McBride and Ross Paterson. Applicative programming with effects. *J. Funct. Programming*, 18(1):1–13, 2008. doi:10.1017/S0956796807006326.

38 Stefan Milius and Tadeusz Litak. Guard your daggers and traces: Properties of guarded (co-)recursion. *Fundamenta Informaticae*, 150:407–449, 2017. special issue FiCS'13. URL: http://arxiv.org/abs/1603.05214, doi:10.3233/FI-2017-1475.

39 Chetan R. Murthy. An evaluation semantics for classical proofs. In Giles Kahn, editor, *Proceedings of LiCS*, pages 96–107. IEEE Computer Society Press, July 1991.

40 Hiroshi Nakano. A modality for recursion. In *Proceedings of LiCS*, pages 255–266. IEEE, 2000.

41 Hiroshi Nakano. Fixed-point logic with the approximation modality and its Kripke completeness. In Naoki Kobayashi and Benjamin C. Pierce, editors, *Proceedings of TACS*, volume 2215 of *LNCS*, pages 165–182. Springer, 2001.

42 Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *B. Symb. Log.*, 5(2):215–244, 1999. URL: http://www.jstor.org/stable/421090.

43 Hiroakira Ono. Glivenko theorems revisited. *Ann. Pure Appl. Logic*, 161(2):246–250, 2009. doi:10.1016/j.apal.2009.05.006.

44 Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *MSCS*, 11(4):511–540, 2001.

45 David J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Appl. Log. Ser.* Kluwer Academic Publishers, 2002.

46 Alex K. Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic*. phdthesis, University of Edinburgh, 1994. URL: http://homepages.inf.ed.ac.uk/als/Research/thesis.ps.gz.

47 Bolesław Sobociński. A theorem concerning a restricted rule of substitution in the field of propositional calculi. I and II. *Notre Dame J. Form. L.*, 15(3 and 4):465–476 and 589–597, 1974. doi:10.1305/ndjfl/1093891408.

48 Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, volume 149 of *Stud. Logic Found. Math.* Elsevier Science Inc., 2006.

49 Vladimir Sotirov. Modal theories with intuitionistic logic. In *Mathematical Logic, Proc. Conf. Math. Logic Dedicated to the Memory of A. A. Markov (1903–1979), Sofia, September 22-23, 1980*, pages 139–171, 1984.

50 Anne S. Troelstra. Marginalia on sequent calculi. *Stud. Logica*, 62(2):291–303, 1999. doi:10.1023/A:1026413320413.

51 Anne S. Troelstra and H. Schwichtenberg. *Basic proof theory*. Number 43 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2000.

52 Anne S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics: An Introduction*, volume 121 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1988.

53 Johan van Benthem, Nick Bezhanishvili, and Wesley Halcrow Holliday. A bimodal perspective on possibility semantics. *J. Log. Comput.*, 2016. URL: http://www.escholarship.org/uc/item/2h5069pq.

54 Albert Visser. On the completeness principle: A study of provability in Heyting's arithmetic and extensions. *Ann. Math. Logic*, 22(3):263–295, 1982.

55 Albert Visser. Closed fragments of provability logics of constructive theories. *J. Symb. Logic*, 73(3):1081–1096, 2008.

56 Frank Wolter and Michael Zakharyaschev. On the relation between intuitionistic and classical modal logics. *Algebra and Logic*, 36:121–125, 1997.

57 Frank Wolter and Michael Zakharyaschev. Intuitionistic modal logics as fragments of classical bimodal logics. In Ewa Orlowska, editor, *Logic at Work, Essays in honour of Helena Rasiowa*, pages 168–186. Springer–Verlag, 1998.

58 Frank Wolter and Michael Zakharyaschev. Modal decision problems. *Studies in Logic and Practical Reasoning*, 3:427–489, 2007. doi:10.1016/S1570-2464(07)80010-3.

59 Michael Zakharyaschev, Frank Wolter, and Alexander Chagrov. Advanced modal logic. In D. M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, pages 83–266. Springer Netherlands, 2001. doi:10.1007/978-94-017-0454-0_2.

Models of Type Theory Based on Moore Paths

Ian Orton[1] **and Andrew M. Pitts**[2]

1 **University of Cambridge Computer Laboratory, Cambridge, UK**
Ian.Orton@cl.cam.ac.uk
2 **University of Cambridge Computer Laboratory, Cambridge, UK**
Andrew.Pitts@cl.cam.ac.uk

Abstract

This paper introduces a new family of models of intensional Martin-Löf type theory. We use constructive ordered algebra in toposes. Identity types in the models are given by a notion of Moore path. By considering a particular gros topos, we show that there is such a model that is *non-truncated*, i.e. contains non-trivial structure at all dimensions. In other words, in this model a type in a nested sequence of identity types can contain more than one element, no matter how great the degree of nesting. Although inspired by existing non-truncated models of type theory based on simplicial and on cubical sets, the notion of model presented here is notable for avoiding any form of Kan filling condition in the semantics of types.

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.3.2 Semantics of Programming Languages

Keywords and phrases dependent type theory, homotopy theory, Moore path, topos

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.28

1 Introduction

Homotopy Type Theory [30] has re-invigorated the study of the intensional version of Martin-Löf type theory [21]. On the one hand, the language of type theory helps to express synthetic constructions and arguments in homotopy theory and higher-dimensional category theory. On the other hand, the geometric and algebraic insights of those highly developed branches of mathematics shed new light on logical and type-theoretic notions. One might say that the familiar *propositions-as-types* analogy has been extended to *propositions-as-types-as-spaces*. In particular, under this analogy there is a *path-oriented* view of intensional (i.e. proof-relevant) equality: proofs of equality of two elements of a type $x, y : A$ correspond to elements of a Martin-Löf identity type $\mathtt{Id}_A\, x\, y$ and behave analogously to paths between two points x, y in a space A. The complicated internal structure of intensional identity types relates to the homotopy classes of path spaces. To make the analogy precise and to exploit it, it helps to have a wide range of models of intensional type theory that embody this path-oriented view of equality in some way. This paper introduces a new family of such models, constructed from *Moore paths* [24] in toposes.

Let $\mathbb{R}_+ = \{r \in \mathbb{R} \mid r \geq 0\}$ be the real half-line with the usual topology. Classically, a Moore path between points x and y in a topological space X is a pair $p = (f, r)$ where $r \in \mathbb{R}_+$ and $f : \mathbb{R}_+ \to X$ is a continuous function with $f\,0 = x$ and $f\,r' = y$ for all $r' \geq r$. We will write $x \sim y$ for the set of Moore paths from x to y, with X understood from the context. Clearly there is a Moore path from x to y iff there is a conventional path, that is, a continuous function $f : [0, 1] \to X$ with $f\,0 = x$ and $f\,1 = y$. The advantage of Moore paths is that they admit degeneracy and composition operations that are unitary and associative

up to equality; whereas for conventional paths these identities only hold up to homotopy. Specifically, given $p = (f, r) \in x \sim y$ and $q = (g, s) \in y \sim z$ we get

$$\mathtt{idp}\, x :\equiv (\lambda\_ \to x, 0) \in x \sim x \qquad\qquad q \bullet p :\equiv (g \bullet^r f, r + s) \in x \sim z \tag{1}$$

where $(g \bullet^r f)\, t$ is equal to $f\, t$ if $t \leq r$ and otherwise is equal to $g(t - r)$. These definitions satisfy $p \bullet \mathtt{idp}\, x = p = \mathtt{idp}\, y \bullet p$ and $r \bullet (q \bullet p) = (r \bullet q) \bullet p$.

We abstract from $\mathbb{R}$ some simple order-algebraic structure [4, chapter VI] sufficient for definition (1) to work in a constructive algebraic setting, rather than a classical topological one; initially the structure of an *ordered abelian group* in some topos suffices and then we extend that to an *ordered commutative ring* to make the models satisfy function extensionality. We use this structure in a topos to develop a model of intensional Martin-Löf type theory with: identity types given by Moore paths, Σ-types, Π-types satisfying function extensionality and inductive types (we just consider disjoint unions and W-types). The existence and properties of type-theoretic universes in these models is left for future work. By considering a particular *gros* topos [10] we get a *non-truncated* instance of our model construction, in other words one where iterated identity types $\mathtt{Id}_A, \mathtt{Id}_{\mathtt{Id}_A}, \mathtt{Id}_{\mathtt{Id}_{\mathtt{Id}_A}}, \ldots$ can be non-trivial to any depth of iteration.

The observation that the strictly associative and unitary nature of composition of Moore paths aids in the interpretation of Martin-Löf identity types is not new; see for example [32, sections 5.1 and 5.5]. However, existing non-truncated models of type theory typically make use of some form of Kan filling condition [15] to cut down to a class of *fibrant* families of types with respect to which path types behave as identity types. Perhaps the main contribution of this paper is to show that one can avoid any form of Kan filling and still get a non-truncated model of intensional Martin-Löf type theory. Instead we use a simple notion of fibrant family phrased just in terms of the usual operation of transporting elements along equality proofs. As a consequence every type, regarded as a family over the terminal type, is fibrant in our setting. In particular, this means that the interval is a first-class type in our models, something which is not true for existing path-oriented models, such as the classical simplicial [17] and constructive cubical sets [3, 6] models; and constructing universes in our setting will not need proofs of their fibrancy (something which is quite tricky for cubical sets). These simplifications, together with an algebraic calculus of "bounded abstractions" for Moore paths that we present in this paper suggest that there may be a useful extension of Martin-Löf type theory with path types which is analogous to, but simpler than, Cubical Type Theory [6]. This is something we plan to look at in future.

Informal type theory

The new models of type theory we present are given in terms of *categories with families* (CwF) [8, 11]. Specifically, we start with an arbitrary topos $\mathcal{E}$, to which can be associated a CwF, for example as in [18, 2]. We write $\mathcal{E}(\Gamma)$ for the set of families indexed by an object $\Gamma \in \mathcal{E}$ and $\mathcal{E}(\Gamma \vdash A)$ for the set of elements of a family $A \in \mathcal{E}(\Gamma)$. One can make $\mathcal{E}(\Gamma)$ into a category whose morphisms between two families $A, B \in \mathcal{E}(\Gamma)$ are elements in $\mathcal{E}(\Gamma.A \vdash B)$, where $\Gamma.A$ is the comprehension object associated with A. The category $\mathcal{E}(\Gamma)$ is equivalent to the slice category $\mathcal{E}/\Gamma$, the equivalence being given on objects by sending families $A \in \mathcal{E}(\Gamma)$ to corresponding projection morphisms $\Gamma.A \to \Gamma$.

We then construct a new CwF by considering families in $\mathcal{E}$ equipped with certain extra structure (the *transport-along-paths* structure of section 3); the elements of a family in the new CwF are just those of the underlying family in $\mathcal{E}$. One could describe this construction using the language of category theory. Instead, as in [26] we find it clearer to express the

construction using an internal language for the CwF associated with $\mathcal{E}$ – a combination of higher-order predicate logic and extensional type theory, with an impredicative universe of propositions given by the subobject classifier in $\mathcal{E}$; see [20]. This use of internal language allows us to give an appealingly simple description of the type constructs in the new CwF. In the text we use this language informally (analogously to the way that [30] develops Homotopy Type Theory); in particular the typing contexts of the judgements in the formal version, such as $[x_0 : A_0, x_1 : A_1(x_0), x_2 : A_2(x_0, x_1)]$, become part of the running text in phrases like "given $x_0 : A_0$, $x_1 : A_1(x_0)$ and $x_2 : A_2(x_0, x_1)$, then…". The arguments we give in sections 2–5 are all constructively valid and in fact do not require the impredicative aspects of topos theory; indeed we have used Agda [1] (with uniqueness of identity proofs and postulates for quotient types) as a tool to develop and to formalise the material presented in those sections. The specific model presented in section 6 uses topological spaces within classical mathematics.

2 Moore Paths in a Topos

Let $\mathcal{E}$ be a topos [19, 14] (which we always assume has a natural number object). A *total order* on an object $\mathtt{R} \in \mathcal{E}$ is given by a subobject $\_ \leqslant \_ \rightarrowtail \mathtt{R} \times \mathtt{R}$ which is not only reflexive, transitive and anti-symmetric, but also satisfies

$$(\forall i, j : \mathtt{R})\ i \leqslant j \ \vee\ j \leqslant i\,. \tag{2}$$

An *ordered abelian group* [4, chapter VI] object in $\mathcal{E}$ is given by $\mathtt{R} \in \mathcal{E}$ plus morphisms $\mathtt{0} : 1 \to \mathtt{R}$, $\mathtt{-} : \mathtt{R} \to \mathtt{R}$ and $\_ + \_ : \mathtt{R} \times \mathtt{R} \to \mathtt{R}$ satisfying the usual abelian group axioms, plus a total order that is respected by addition:

$$(\forall i, j, k : \mathtt{R})\ i \leqslant j \ \Rightarrow\ k + i \leqslant k + j\,. \tag{3}$$

We develop properties of Moore paths in $\mathcal{E}$ with respect to such an object $\mathtt{R}$.

Note that since $\mathcal{E}$ may not be Boolean, we do not necessarily have the trichotomy property $(\forall i, j : \mathtt{R})\ i < j \ \vee\ i = j \ \vee\ j < i$ for the associated relation $i < j \ :\equiv \neg(j \leqslant i)$. So when defining functions by cases using (2) we have to verify that the clauses for $i \leqslant j$ and for $j \leqslant i$ agree on the overlap, where $i = j$ by antisymmetry. For example, the *half-line*

$$\mathtt{R_+} :\equiv \{i : \mathtt{R} \mid \mathtt{0} \leqslant i\} \tag{4}$$

associated with $\mathtt{R}$ has binary operations $\mathtt{R_+} \times \mathtt{R_+} \to \mathtt{R_+}$ of *truncated subtraction* $\_ \dot{-} \_$ and *minimum* $\mathtt{min}$ well-defined as follows (where, as usual, we write $i + (\mathtt{-}\, j)$ as $i - j$):

$$(\forall i, j : \mathtt{R_+})\ \begin{array}{lllll} i \leqslant j & \Rightarrow & i \dot{-} j = \mathtt{0} & \wedge & \mathtt{min}(i, j) = i \\ j \leqslant i & \Rightarrow & i \dot{-} j = i - j & \wedge & \mathtt{min}(i, j) = j \end{array} \tag{5}$$

▶ **Definition 2.1** (Moore path objects)**.** For each object $\Gamma \in \mathcal{E}$, the family $(x \sim y \mid x, y : \Gamma) \in \mathcal{E}(\Gamma \times \Gamma)$ of *Moore path objects* is defined by:

$$x \sim y \ :\equiv\ \{(f, i) : (\mathtt{R_+} \to \Gamma) \times \mathtt{R_+} \mid f\,\mathtt{0} = x \ \wedge\ (\forall j \geqslant i)\ f\, j = y\}\,. \tag{6}$$

We use the following notation:

$$\begin{array}{ll} \mathsf{|}\_\mathsf{|} : (x \sim y) \to \mathtt{R_+} & \_\,@\,\_ : (x \sim y) \to \mathtt{R_+} \to \Gamma \\ \mathsf{|}(f, i)\mathsf{|} = i & (f, i)\,@\, j = f\, j \end{array} \tag{7}$$

Following [5, Section 2], we call $\mathtt{|}p\mathtt{|}$ the *shape* of the path $p : x \sim y$. Thus if $p : x \sim y$, then $p\,\mathtt{0} = x$ and $p\,\mathtt{@}\,\mathtt{|}p\mathtt{|} = y$. *From now on we will just write* $p\,\mathtt{@}\,i$ *as* $p\,i$.

Given $\gamma : \Gamma \to \Delta$ in $\mathcal{E}$, there is a congruence operation mapping paths in Γ to those in Δ:

$$\gamma\ \mathtt{'}\ : (x \sim y) \to (\gamma\,x \sim \gamma\,y) \qquad\qquad \gamma\ \mathtt{'}\ (f, i) :\equiv (\gamma \circ f, i) \tag{8}$$

▶ **Lemma 2.2** (degenerate and composite paths)**.** *For each* $\Gamma \in \mathcal{E}$*, given* $x, y, z : \Gamma$*,* $p : x \sim y$ *and* $q : y \sim z$*, there are* degenerate paths $\mathtt{idp}\,x : x \sim x$ *and* composite paths $q \bullet p : x \sim z$ *well-defined by the requirements*

$$\mathtt{|idp}\,x\mathtt{|} = \mathtt{0} \quad \wedge \quad (\forall i : \mathtt{R_+})\ (\mathtt{idp}\,x)\,i = x$$

$$\mathtt{|}q \bullet p\mathtt{|} = \mathtt{|}p\mathtt{|} + \mathtt{|}q\mathtt{|} \quad \wedge \quad (\forall i : \mathtt{R_+}) \begin{array}{lcl} i \leqslant \mathtt{|}p\mathtt{|} & \Rightarrow & (q \bullet p)\,i = p\,i \\ \mathtt{|}p\mathtt{|} \leqslant i & \Rightarrow & (q \bullet p)\,i = q\,(i - \mathtt{|}p\mathtt{|}) \end{array}$$

and satisfying $p \bullet (\mathtt{idp}\,x) = p = (\mathtt{idp}\,y) \bullet p$*,* $(r \bullet q) \bullet p = r \bullet (q \bullet p)$*,* $\gamma\ \mathtt{'}\ (\mathtt{idp}\,x) = \mathtt{idp}(\gamma\,x)$ *and* $\gamma\ \mathtt{'}\ (q \bullet p) = (\gamma\ \mathtt{'}\ q) \bullet (\gamma\ \mathtt{'}\ p)$*.*

Proof. One just has to check that the usual definitions for classical Moore paths are constructive and only make use of the ordered abelian group properties of the reals. ◀

▶ **Lemma 2.3** (path reversal)**.** *For each* $\Gamma \in \mathcal{E}$*, given* $x, y : \Gamma$ *and* $p : x \sim y$*, there is a* reversed path $\mathtt{rev}\,p : y \sim x$ *well-defined by the requirements* $\mathtt{|rev}\,p\mathtt{|} = \mathtt{|}p\mathtt{|}$ *and* $(\forall i : \mathtt{R_+})\ (\mathtt{rev}\,p)\,i = p\,(\mathtt{|}p\mathtt{|} \dot{-} i)$ *(where* $\dot{-}$ *is as in* (5)*) and satisfying* $\mathtt{rev}(\mathtt{idp}\,x) = \mathtt{idp}\,x$*,* $\mathtt{rev}(q \bullet p) = (\mathtt{rev}\,p) \bullet (\mathtt{rev}\,q)$*,* $\mathtt{rev}(\mathtt{rev}\,p) = p$ *and* $\gamma\ \mathtt{'}\ (\mathtt{rev}\,p) = \mathtt{rev}(\gamma\ \mathtt{'}\ p)$*.*

Proof. As for the previous lemma, this follows straightforwardly from the axioms of ordered abelian groups within constructive logic. ◀

Although the definition of $\mathtt{rev}\,p$ is standard, the above equational properties are not often mentioned in the literature. However, they are crucial for the construction of identity types in section 3 to work. What usually gets a mention is the fact that up to homotopy $\mathtt{rev}\,p$ is a two-sided inverse for p with respect to $\bullet$. Paths $(\mathtt{rev}\,p) \bullet p \sim \mathtt{idp}\,x$ and $p \bullet (\mathtt{rev}\,p) \sim \mathtt{idp}\,y$ can be constructed using the path contraction operation given below; we do not bother to do that, because they are also a consequence of the *path induction* [30, 1.12.1] property of identity types that follows from Theorem 3.9.

▶ **Definition 2.4** (bounded abstractions)**.** The following binding syntax is very convenient for describing Moore paths. For each $\Gamma \in \mathcal{E}$, if $\lambda i \to \varphi(i)$ describes a function in $\mathtt{R_+} \to \Gamma$, then for each $j : \mathtt{R_+}$ using the $\mathtt{min}$ function (5) we get a path $\langle i \leqslant j \rangle \varphi(i) : \varphi(\mathtt{0}) \sim \varphi(j)$ in Γ by defining:

$$\langle i \leqslant j \rangle \varphi(i) :\equiv (\lambda i \to \varphi(\mathtt{min}(i, j))\ , j) \tag{9}$$

(i is bound in the above expression). Note that

$$\gamma\ \mathtt{'}\ \langle i \leqslant j \rangle \varphi(i) = \langle i \leqslant j \rangle \gamma(\varphi(i))\,, \tag{10}$$

$$\langle i \leqslant \mathtt{0} \rangle \varphi(i) = \mathtt{idp}(\varphi(\mathtt{0}))\,, \tag{11}$$

$$\langle i \leqslant \mathtt{|}p\mathtt{|} \rangle (p\,i) = p\,. \tag{12}$$

▶ **Lemma 2.5** (path contraction)**.** *Given* $\Gamma \in \mathcal{E}$*, for any path* $p : x \sim y$ *in* Γ *and* $i : \mathtt{R_+}$*, there is a path* $p_{\leqslant i} : x \sim p\,i$ *satisfying* $p_{\leqslant \mathtt{0}} = \mathtt{idp}\,x$ *and* $(\forall i \geqslant \mathtt{|}p\mathtt{|})\ p_{\leqslant i} = p$*.*

Proof. Using the bounded abstraction notation and the `min` function (5), we define

$$p_{\leqslant i} :\equiv \langle j \leqslant \mathtt{min}(|p|, i)\rangle(p\, j)\,. \tag{13}$$

Since $p\,(\mathtt{min}(|p|, i)) = p\, i$, this does give a path $x \sim p\, i$; and it has the required properties by (11) and (12). ◀

Using $p_{\leqslant i}$ we get for each $x : \Gamma$ that $\sum_{y:\Gamma} x \sim y$ is path-contractible [30, section 3.11] with centre $(x, \mathtt{idp}\, x)$, since for each $y : \Gamma$ and $p : x \sim y$ we have a path $\langle i \leqslant |p|\rangle(p\, i, p_{\leqslant i})$ in $(x, \mathtt{idp}\, x) \sim (y, p)$. This is part of the more general fact that Moore paths model identity types, which we show in the next section.

3 Transport along Paths

In this section we continue with the assumptions of the previous one: $\mathcal{E}$ is a topos (with associated CwF) containing an ordered abelian group object $\mathtt{R}$. We want objects of Moore paths with respect to $\mathtt{R}$ (Definition 2.1) to give a model of identity types, as well as other type formers. Recall that in Martin-Löf type theory elements of identity types $\mathtt{Id}_\Gamma\, x\, y$ give rise to transport functions $A\, x \to A\, y$ between members of a family of types $(A\, x \mid x : \Gamma)$ over a type Γ; see for example [30, section 2.3]. Therefore, for each object $\Gamma \in \mathcal{E}$, we should restrict attention to families $A \in \mathcal{E}(\Gamma)$ that come equipped at least with some sort of transport operation taking a path $p : x \sim y$ in Γ and an element $a : A\, x$ to an element $p_* a : A\, y$. This leads to the following definition.

▶ **Definition 3.1** (fibrations)**.** Given an object $\Gamma \in \mathcal{E}$, a *transport-along-paths* (tap) structure for a family $A \in \mathcal{E}(\Gamma)$ is a $(\Gamma \times \Gamma)$-indexed family of morphisms $((\_)_* : x \sim y \to (A\, x \to A\, y) \mid x, y : \Gamma)$ satisfying for all $x : \Gamma$ and $a : A\, x$

$$(\mathtt{idp}\, x)_* a = a\,. \tag{14}$$

We write $\mathbf{Fib}(\Gamma)$ for the families over Γ equipped with a tap structure and call them *fibrations*. They are stable under re-indexing: given $\gamma : \Delta \to \Gamma$ in $\mathcal{E}$ and $A \in \mathbf{Fib}(\Gamma)$, then $A\, \gamma :\equiv (A(\gamma\, x) \mid x : \Delta) \in \mathcal{E}(\Delta)$ has a tap structure via the congruence operation (8), taking the transport of $a : (A\, \gamma)\, x = A(\gamma\, x)$ along $p : x \sim y$ to be $(\gamma \mathbin{\texttt{'}} p)_* a : A(\gamma\, y)$. This re-indexing of tap structure respects composition and identities in $\mathcal{E}$. So $\mathbf{Fib}$ inherits the structure of a CwF from that of $\mathcal{E}$, with the set of elements of a fibration being the elements of the underlying family in $\mathcal{E}$, that is $\mathbf{Fib}(\Gamma \vdash A) = \mathcal{E}(\Gamma \vdash A)$.

▶ Remark 3.2. In the above definition we do not require the transport action $\_{}_*{}\_$ to be strictly associative, because that property is not needed to prove that fibrations give a model of type theory. However, we always have associativity up to homotopy, i.e. there is a path $(q \bullet p)_* a \sim q_*(p_* a)$. This is a consequence of the fact that, with respect to the above notion of fibration, Moore paths model identity types (see Theorem 3.9 below) and hence support path induction [30, section 2.9]). This allows one to construct a path $(q \bullet p)_* a \sim q_*(p_* a)$ for any p from the case when $p = \mathtt{idp}\, x$, where one has the degenerate path for $(q \bullet (\mathtt{idp}\, x))_* a = q_* a = q_*((\mathtt{idp}\, x)_* a)$.

We show that the CwF $\mathbf{Fib}$ inherits some type structure from that of $\mathcal{E}$. To begin with, note that every object $\Gamma \in \mathcal{E}$, regarded as a family over the terminal object 1, can be made into a fibration via the trivial transport action $p_* a = a$. So in particular the half-line $\mathtt{R}_+$ is a fibration over 1. Also, it is easy to see that the tap structure on families $A, B \in \mathbf{Fib}(\Gamma)$ lifts to one on the family of disjoint unions $(A\, x + B\, x \mid x : \Gamma)$ (stably under re-indexing) so that

the CwF **Fib** supports disjoint union types. To describe Σ- and Π-types in **Fib** we have to lift paths in Γ to paths in comprehension objects $\Gamma.A$ of fibrations $A \in \mathbf{Fib}(\Gamma)$:

▶ **Lemma 3.3** (path lifting). *Given $\Gamma \in \mathcal{E}$ and $A \in \mathbf{Fib}(\Gamma)$, for each path $p : x \sim y$ in Γ and each $a : A\,x$, there is a path $\mathtt{lift}(p, a) : (x, a) \sim (y, p_* a)$ in $\Gamma.A$ satisfying*

$$\mathtt{lift}(\mathtt{idp}\,x, a) = \mathtt{idp}(x, a) \tag{15}$$

and stable under re-indexing along morphisms $\Delta \to \Gamma$ in $\mathcal{E}$.

Proof. We can use path contraction (Lemma 2.5) to express path lifting using the bounded abstraction notation from Definition 2.4:

$$\mathtt{lift}(p, a) :\equiv \langle i \leqslant |p| \rangle (p\,i, (p_{\leqslant i})_* a)\,. \tag{16}$$

Since the path contraction operation satisfies $p_{\leqslant 0} = \mathtt{idp}\,x$ and $p_{\leqslant |p|} = p$, this does indeed give a path in $(x, a) \sim (y, p_* a)$. The desired properties of `lift` follow from corresponding properties of path contraction. ◀

▶ **Theorem 3.4** (Σ- and Π-types). *Given $\Gamma \in \mathcal{E}$, $A \in \mathbf{Fib}(\Gamma)$ and $B \in \mathbf{Fib}(\Gamma.A)$, the families $\Sigma\,A\,B :\equiv (\sum_{a:A\,x} B(x, a) \mid x : \Gamma)$ and $\Pi\,A\,B :\equiv (\prod_{a:A\,x} B(x, a) \mid x : \Gamma)$ in $\mathcal{E}(\Gamma)$ have tap structures that are stable under re-indexing along morphisms $\Delta \to \Gamma$ in $\mathcal{E}$. Hence the CwF* **Fib** *supports Σ-and Π-types [11, Definitions 3.15 and 3.18].*

Proof. Given a path $p : x \sim y$ in Γ, we get functions $\Sigma\,A\,B\,x \to \Sigma\,A\,B\,y$ and $\Pi\,A\,B\,x \to \Pi\,A\,B\,y$ using path lifting and (in the second case) path reversal:

$$p_*(a, b) :\equiv (p_* a, \mathtt{lift}(p, a)_* b) \in \Sigma\,A\,B\,y \qquad \text{where } (a, b) : \Sigma\,A\,B\,x \tag{17}$$

$$(p_* f)\,a :\equiv (\mathtt{rev}(\mathtt{lift}(\mathtt{rev}\,p, a)))_* f((\mathtt{rev}\,p)_* a) \in B(y, a) \qquad \begin{array}{l}\text{where } f : \Pi\,A\,B\,x \\ \text{and } a : A\,y\end{array} \tag{18}$$

The properties of path lifting given in Lemma 3.3 suffice to show that (17) inherits the properties of a tap structure from those for A and B and is stable under re-indexing. The same is true for (18) except that one also has to use the properties of path reversal given in Lemma 2.3. ◀

Since we only consider toposes with a natural number object, the CwF associated with $\mathcal{E}$ can interpret types of well-founded trees (W-types) [30, section 5.3]; see [23, Propositions 3.6 and 3.8]. We write $\mathtt{W}_{x:A}\,B\,x$ for the object of well-founded trees determined by a family $B \in \mathcal{E}(A)$, with constructor $\mathtt{sup} : \sum_{y:A}(B\,y \to \mathtt{W}_{x:A}\,B\,x) \to \mathtt{W}_{x:A}\,B\,x$.

▶ **Theorem 3.5** (W-types). *Given $A \in \mathbf{Fib}(\Gamma)$ and $B \in \mathbf{Fib}(\Gamma.A)$, the family*

$$W\,A\,B :\equiv (\mathtt{W}_{a:A\,x} B(x, a) \mid x : \Gamma) \in \mathcal{E}(\Gamma)$$

has a tap structure that is stable under re-indexing along morphisms $\Delta \to \Gamma$ in $\mathcal{E}$. Therefore the CwF **Fib** *supports W-types.*

Proof. Given a path $p : x \sim y$ in Γ, we get a function $W\,A\,B\,x \to W\,A\,B\,y$ via the following well-founded recursion equation:

$$p_*\,\mathtt{sup}(a\,, f) = \mathtt{sup}\,(p_* a\,, \lambda b \to p_* f((\mathtt{rev}(\mathtt{lift}(p, a)))_* b)) \qquad \begin{array}{l}\text{where } a : A\,x \text{ and} \\ f : B(x, a) \to W\,A\,B\,x\end{array} \tag{19}$$

Well-founded inductions using the properties of reversal and lifting given in Lemmas 2.3 and 3.3 suffice to show that this inherits the properties of a tap structure from those for A and B and is stable under re-indexing. ◀

So far we have lifted some type structure from the CwF associated with $\mathcal{E}$ to the CwF **Fib**. Since $\mathcal{E}$ is a topos, it certainly has *extensional* identity types [22], inhabitation of which coincides with judgemental equality, and those could be lifted to **Fib**. However, we wish to show that Moore path objects become *intensional* identity types in **Fib**. We will use Hofmann's version [11] of the structure in a CwF needed to model such types. To do so, let us fix some notation for a CwF **C**. Re-indexing of a family $A \in \mathbf{C}(\Gamma)$ and an element $\alpha \in \mathbf{C}(\Gamma \vdash A)$ along a morphism $\gamma : \Delta \to \Gamma$ will just be denoted by $A\,\gamma \in \mathbf{C}(\Delta)$ and $\alpha\,\gamma \in \mathbf{C}(\Delta \vdash A\,\gamma)$; and given an element $\beta \in \mathbf{C}(\Delta \vdash A\,\gamma)$, then $\langle \gamma\,, \beta \rangle$ denotes the unique morphism $\Delta \to \Gamma.A$ whose composition with $\mathtt{fst} : \Gamma.A \to \Gamma$ is γ and whose re-indexing of the generic element $\mathtt{snd} \in \mathbf{C}(\Gamma.A \vdash A\,\mathtt{fst})$ is β.

▶ **Definition 3.6.** Following Hofmann [11, Definition 3.19], we say that a CwF **C** *supports the interpretation of intensional identity types* if for each object $\Gamma \in \mathbf{C}$ and each family $A \in \mathbf{C}(\Gamma)$ the following data is given and is stable under re-indexing along any $\gamma : \Delta \to \Gamma$:

- a family $\mathtt{Id}_A \in \mathbf{C}(\Gamma.A.A\,\mathtt{fst})$,
- a morphism $\mathtt{Refl}_A : \Gamma.A \to \Gamma.A.A\,\mathtt{fst}.\,\mathtt{Id}_A$ such that $\mathtt{fst} \circ \mathtt{Refl}_A$ equals the diagonal morphism $\langle \mathtt{id}\,, \mathtt{snd} \rangle : \Gamma.A \to \Gamma.A.A\,\mathtt{fst}$,
- a function mapping each $B \in \mathbf{C}(\Gamma.A.A\,\mathtt{fst}.\,\mathtt{Id}_A)$ and $\beta \in \mathbf{C}(\Gamma.A \vdash B\ \mathtt{Refl}_A)$ to an element $\mathtt{J}_A\,B\,\beta \in \mathbf{C}(\Gamma.A.A\,\mathtt{fst}.\,\mathtt{Id}_A \vdash B)$ such that the re-indexing $(\mathtt{J}_A\,B\,\beta)\ \mathtt{Refl}_A$ equals β.

Given an object Γ of the topos $\mathcal{E}$ and a family $A \in \mathcal{E}(\Gamma)$, we can use the family of Moore path objects $\_ \sim \_$ (Definition 2.1) to define a family $\mathtt{Id}_A \in \mathcal{E}(\Gamma.A.A\,\mathtt{fst})$ as follows:

$$\mathtt{Id}_A :\equiv (a_1 \sim a_2 \mid ((x, a_1), a_2) : \Gamma.A.A\,\mathtt{fst})\,. \tag{20}$$

We will show that this together with suitable `Refl` and `J` operations give an instance of Definition 3.6 for the CwF **Fib**. In particular $\mathtt{Id}_A$ has a tap structure when A does. To see this we first need to analyse paths in $\Gamma.A$ in terms of paths in Γ and in the fibres $A\,x$ (for $x : \Gamma$).

▶ **Lemma 3.7.** *Given $\Gamma \in \mathcal{E}$ and $A \in \mathbf{Fib}(\Gamma)$, for each path $p : (x, a) \sim (y, b)$ in $\Gamma.A$, there is a path $\mathtt{snd}(p) : (\mathtt{fst}\ \mathtt{'}\ p)_* a \sim b$ in $A\,y$ satisfying*

$$\mathtt{snd}(\mathtt{idp}(x, a)) = \mathtt{idp}\,a \tag{21}$$

and stable under re-indexing along any $\gamma : \Delta \to \Gamma$ in $\mathcal{E}$.

Proof. We use a reversed version of the path-contraction operation from Lemma 2.5: for each path $q : x \sim y$ and each $i : \mathtt{R}_+$, we define $q_{\geqslant i} : q\,i \sim y$ by

$$q_{\geqslant i} :\equiv \mathtt{rev}\left((\mathtt{rev}\,q)_{\leqslant (|q| \dot{-} i)}\right) \tag{22}$$

where $\dot{-}$ is the truncated subtraction operation from (5). Given $p : (x, a) \sim (y, b)$, then $\mathtt{fst}\ \mathtt{'}\ p : x \sim y$ and hence for each $i : \mathtt{R}_+$ we have $(\mathtt{fst}\ \mathtt{'}\ p)_{\geqslant i} : (\mathtt{fst}\ \mathtt{'}\ p)\,i \sim y$; and since $\mathtt{snd}(p\,i) : A(\mathtt{fst}(p\,i)) = A((\mathtt{fst}\ \mathtt{'}\ p)\,i)$, we get $((\mathtt{fst}\ \mathtt{'}\ p)_{\geqslant i})_*\,\mathtt{snd}(p\,i) : A\,y$. So we can define

$$\mathtt{snd}(p) = \langle i \leqslant |p| \rangle\,(((\mathtt{fst}\ \mathtt{'}\ p)_{\geqslant i})_*\,\mathtt{snd}(p\,i)) \tag{23}$$

to get a path in $(\mathtt{fst}\ \mathtt{'}\ p)_* a \sim b$ in $A\,y$. Property (21) and stability under re-indexing follow from (10) and (11). ◀

▶ **Lemma 3.8.** *Given $\Gamma \in \mathcal{E}$ and a fibration $A \in \mathbf{Fib}(\Gamma)$, the family $\mathtt{Id}_A \in \mathcal{E}(\Gamma.A.A\,\mathtt{fst})$ defined in* (20) *has a tap structure that is stable under re-indexing along morphisms $\Delta \to \Gamma$ in $\mathcal{E}$.*

Proof. First note that re-indexing the fibration $A \in \mathbf{Fib}(\Gamma)$ along $\mathtt{fst} : \Gamma.A \to \Gamma$ we get $A\,\mathtt{fst} \in \mathbf{Fib}(\Gamma.A)$. Given a path $p : ((x, a_1), a_2) \sim ((y, b_1), b_2)$ in $\Gamma.A.A\,\mathtt{fst}$, we can apply Lemma 3.7 to the paths $\mathtt{fst}\ \mathtt{'}\ p : (x, a_1) \sim (y, b_1)$ and $\langle \mathtt{fst} \circ \mathtt{fst}\,, \mathtt{snd}\rangle\ \mathtt{'}\ p : (x, a_2) \sim (y, b_2)$ in $\Gamma.A$ to get paths $p_1 :\equiv \mathtt{snd}(\mathtt{fst}\ \mathtt{'}\ p) : (\mathtt{fst}\ \mathtt{'}\ (\mathtt{fst}\ \mathtt{'}\ p))_* a_1 \sim b_1$ and $p_2 :\equiv \mathtt{snd}(\langle \mathtt{fst} \circ \mathtt{fst}\,, \mathtt{snd}\rangle\ \mathtt{'}\ p) : (\mathtt{fst}\ \mathtt{'}\ (\mathtt{fst}\ \mathtt{'}\ p))_* a_2 \sim b_2$ in $A\,y$ (using the fact that $\mathtt{fst} \circ \langle \mathtt{fst} \circ \mathtt{fst}\,, \mathtt{snd}\rangle = \mathtt{fst} \circ \mathtt{fst}$). Thus for each path $q : a_1 \sim a_2$ in $A\,x$, we can compose together the paths

$$b_1 \xrightarrow{\mathtt{rev}(p_1)} (\mathtt{fst}\ \mathtt{'}\ (\mathtt{fst}\ \mathtt{'}\ p))_* a_1 \xrightarrow{((\mathtt{fst}\mathtt{'}(\mathtt{fst}\mathtt{'}p))_*)\mathtt{'}q} (\mathtt{fst}\ \mathtt{'}\ (\mathtt{fst}\ \mathtt{'}\ p))_* a_2 \xrightarrow{p_2} b_2$$

in $A\,y$ to get a path $b_1 \sim b_2$. So we get a function $p_* : \mathtt{Id}_A((x, a_1), a_2) \to \mathtt{Id}_A((y, b_1), b_2)$ defined by:

$$p_* q = \mathtt{snd}(\langle \mathtt{fst} \circ \mathtt{fst}\,, \mathtt{snd}\rangle\ \mathtt{'}\ p) \bullet (((\mathtt{fst}\ \mathtt{'}\ (\mathtt{fst}\ \mathtt{'}\ p))_*)\ \mathtt{'}\ q) \bullet \mathtt{rev}(\mathtt{snd}(\mathtt{fst}\ \mathtt{'}\ p))\,. \tag{24}$$

The properties of Moore path reversal (Lemma 2.3) together with Lemma 3.7 suffice to show that this definition inherits the property of a tap structure (14) from the one for A and that it is stable under re-indexing. ◀

▶ **Theorem 3.9** (identity types)**.** *The CwF* **Fib** *of Definition 3.1 supports the interpretation of intensional identity types (Definition 3.6), given by Moore path objects as in* (20)*.*

Proof. In view of Lemma 3.8, it just remains to define the `Refl` and `J` operations as in Definition 3.6. Given $\Gamma \in \mathcal{E}$ and $A \in \mathbf{Fib}(\Gamma)$, we get $\mathtt{Refl}_A : \Gamma.A \to \Gamma.A.A\,\mathtt{fst}.\,\mathtt{Id}_A$ by defining

$$\mathtt{Refl}_A(x, a) :\equiv (((x, a), a), \mathtt{idp}\,a)\,. \tag{25}$$

Note that $(\mathtt{fst} \circ \mathtt{Refl}_A)(x, a) = ((x, a), a) = \langle \mathtt{id}\,, \mathtt{snd}\rangle(x, a)$, as required.

Given $B \in \mathbf{Fib}(\Gamma.A.A\,\mathtt{fst}.\,\mathtt{Id}_A)$ and $\beta \in \mathcal{E}(\Gamma.A \vdash B\ \mathtt{Refl}_A)$, for each path $p : a_1 \sim a_2$ in $A\,x$ using Lemma 2.5 we have the following path in $\Gamma.A.A\,\mathtt{fst}.\,\mathtt{Id}_A$:

$$\langle i \leqslant |p| \rangle (((x, a_1), p\,i), p_{\leqslant i}) : (((x, a_1), a_1), \mathtt{idp}\,a_1) \sim (((x, a_1), a_2), p)\,.$$

Since B has a tap structure we can transport $\beta(x, a_1) : B(((x, a_1), a_1), \mathtt{idp}\,a_1)$ along this path to get an element of $B(((x, a_1), a_2), p)$. So we can define $\mathtt{J}_A\,B\,\beta \in \mathcal{E}(\Gamma.A.A\,\mathtt{fst}.\,\mathtt{Id}_A \vdash B)$ by:

$$\mathtt{J}_A\,B\,\beta\,(((x, a_1), a_2), p) :\equiv (\langle i \leqslant |p| \rangle (((x, a_1), p\,i), p_{\leqslant i}))_*\,\beta(x, a_1)\,. \tag{26}$$

`J` has the required computation property, because

$$\begin{aligned}
((\mathtt{J}_A\,B\,\beta)\,\mathtt{Refl}_A)(x, a) &= \mathtt{J}_A\,B\,\beta\,(((x, a), a), \mathtt{idp}\,a) && \text{by (25)}\\
&= (\langle i \leqslant 0 \rangle (((x, a), a)\,\mathtt{idp}\,a))_*\,\beta(x, a) && \text{by (26) and (13)}\\
&= \mathtt{idp}(((x, a), a)\,\mathtt{idp}\,a)_*\beta(x, a) && \text{by (11)}\\
&= \beta(x, a) && \text{by (14)}
\end{aligned}$$

Stability of `Refl` under re-indexing follows from the fact that the congruence operations $\gamma\ \mathtt{'}$ preserve degenerate paths; and stability of `J` uses (10) and the fact that the path contraction operation $(\_)_{\leqslant i}$ is preserved by the congruence operations $\gamma\ \mathtt{'}$. ◀

4 Function Extensionality

In this section we investigate whether functions in the CwF **Fib** behave extensionally with respect to identity types given by Moore paths (Theorem 3.9). So far we have needed only rather weak assumptions about the object $\mathtt{R}$, namely that it is an ordered abelian group. For function extensionality to hold, we need to assume more. To see why, consider the constant function $\mathtt{k}_0 = \lambda i \to \mathtt{0} : \mathtt{R}_+ \to \mathtt{R}_+$ and the identity function $\mathtt{id} : \mathtt{R}_+ \to \mathtt{R}_+$.[1] If equality means existence of a Moore path, then these functions are extensionally equal, because we have $\lambda i \to \langle j \leqslant i \rangle j : \prod_{i:\mathtt{R}_+} (\mathtt{k}_0\, i \sim \mathtt{id}\, i)$. So if the principle of function extensionality is to hold with respect to Moore paths, then there will have to be a path $p : \mathtt{k}_0 \sim \mathtt{id}$ in $\mathtt{R}_+ \to \mathtt{R}_+$. What is its shape $|p|$? Since p does not depend upon any assumptions, $|p|$ would have to be a global element $1 \to \mathtt{R}$ in the topos $\mathcal{E}$. Our current assumptions about $\mathtt{R}$ only guarantee the existence of one such global element, namely $\mathtt{0}$. However, if $|p| = \mathtt{0}$ then $\mathtt{k}_0 = \mathtt{id}$, that is $(\forall i : \mathtt{R}_+)\ \mathtt{0} = i$, and the model of type theory is degenerate. So we need some extra assumptions about $\mathtt{R}$ if function extensionality is to hold without collapsing everything. We do not attempt to find the minimal such assumptions, but rather identify some well-known part of (constructive, ordered) Algebra [4, chapter VI] that does the job, in the expectation that this makes easier the task of finding specific models with good properties (which we address in section 6).

▶ **Definition 4.1** (order-ringed topos)**.** An *order-ringed topos* $(\mathcal{E}, \mathtt{R})$ is a topos $\mathcal{E}$ (with an associated CwF) together with an *ordered commutative ring object* $\mathtt{R}$ in $\mathcal{E}$. Thus as well as an ordered abelian group structure (see section 2), $\mathtt{R}$ has a unit $\mathtt{1} : 1 \to \mathtt{R}$ and a multiplication $\_ \cdot \_ : \mathtt{R} \times \mathtt{R} \to \mathtt{R}$ satisfying the usual axioms for a commutative ring; furthermore multiplication respects the order relation in the sense that

$$(\forall i, j : \mathtt{R})\ \mathtt{0} \leqslant i \ \wedge\ \mathtt{0} \leqslant j \ \Rightarrow\ \mathtt{0} \leqslant i \cdot j\,. \tag{27}$$

From now on we assume that we are given such an order-ringed topos $(\mathcal{E}, \mathtt{R})$.

Given an object $\Gamma \in \mathcal{E}$ and a family $A \in \mathcal{E}(\Gamma)$, if $p : f \sim g$ is a path between dependent functions $f, g : \prod_{x:\Gamma} A\,x$, then for each $x : \Gamma$ we can apply the path congruence operation (8) to $\lambda f \to f\,x : (\prod_{x:\Gamma} A\,x) \to A\,x$ and p to obtain a path $(\lambda f \to f\,x)\ \mathtt{'}\ p : f\,x \sim g\,x$ in $A\,x$. This gives us the following function (which coincides with the canonical function obtained by path induction as in [30, section 2.9]):

$$\begin{aligned} &\mathtt{happly} : (f \sim g) \to \textstyle\prod_{x:\Gamma}(f\,x \sim g\,x) \\ &\mathtt{happly}\ p\,x :\equiv (\lambda f \to f\,x)\ \mathtt{'}\ p \end{aligned} \qquad (\text{where } f, g : \textstyle\prod_{x:\Gamma} A\,x) \tag{28}$$

▶ **Theorem 4.2** (function extensionality modulo $\sim$)**.** *Given an order-ringed topos* $(\mathcal{E}, \mathtt{R})$*, an object* $\Gamma \in \mathcal{E}$ *and a family* $A \in \mathcal{E}(\Gamma)$*, for all* $f, g : \prod_{x:\Gamma} A\,x$ *there is a function* $\mathtt{funext} : (\prod_{x:\Gamma}(f\,x \sim g\,x)) \to (f \sim g)$*. Furthermore this function is quasi-inverse [30, Definition 2.4.6] to* $\mathtt{happly}$*, that is, that for all* $e : \prod_{x:\Gamma}(f\,x \sim g\,x)$ *and* $p : f \sim g$ *there are paths* $\varepsilon\, e : \mathtt{happly}(\mathtt{funext}\, e) \sim e$ *in* $\prod_{x:\Gamma}(f\,x \sim g\,x)$ *and* $\eta\, p : p \sim \mathtt{funext}(\mathtt{happly}\, p)$ *in* $f \sim g$*.*

Proof. If $e : \prod_{x:\Gamma}(f\,x \sim g\,x)$, then for all $x : \Gamma$ we have $e\,x\,(\mathtt{0} \cdot |e\,x|) = e\,x\,\mathtt{0} = f\,x$ and $e\,x\,(\mathtt{1} \cdot |e\,x|) = e\,x\,|e\,x| = g\,x$. So there is a path $\mathtt{funext}\, e : f \sim g$ in $\prod_{x:\Gamma} A\,x$ given by:

$$\mathtt{funext}\, e = \langle i \leqslant \mathtt{1} \rangle \lambda x \to e\,x\,(i \cdot |e\,x|)\,. \tag{29}$$

[1] Function extensionality for **Fib** only concerns functions between fibrant families; but recall that in **Fib** all objects, and in particular $\mathtt{R}_+$, are fibrant as trivial families over the terminal object.

Note that by (10) we have for each $x : \Gamma$

$$\begin{aligned}\mathtt{happly}(\mathtt{funext}\, e)\, x &= (\lambda f \to f\, x)\ \mathbin{\text{'}}\ \langle i \leqslant \mathtt{1}\rangle \lambda x \to e\, x\, (i \cdot |e\, x|) \\ &= \langle i \leqslant \mathtt{1}\rangle (e\, x\, (i \cdot |e\, x|))\end{aligned}$$

whereas $e\, x = \langle i \leqslant |e\, x|\rangle (e\, x\, i)$ by (12). So to get a path from $\mathtt{happly}(\mathtt{funext}\, e)\, x$ to $e\, x$ we need to interpolate shapes from $\mathtt{1}$ to $|e\, x|$ while at the same time interpolating the argument of $e\, x\, \_$ from $i \cdot |e\, x|$ to $i \cdot \mathtt{1} = i$. So for each $j : \mathtt{R}$ with $\mathtt{0} \leqslant j \leqslant \mathtt{1}$, consider

$$u_{j,x} :\equiv \mathtt{1} - j + j \cdot |e\, x| \quad \text{and} \quad v_{j,x} :\equiv (\mathtt{1} - j) \cdot |e\, x| + j\,.$$

Calculating with the ring axioms we find that $u_{j,x} \cdot v_{j,x} = |e\, x| + j \cdot (\mathtt{1} - j) \cdot (|e\, x| - \mathtt{1})^2$. Since $\mathtt{0} \leqslant j$ and $\mathtt{0} \leqslant \mathtt{1} - j$ by assumption and since squares are always positive in an ordered ring [4, VI.19], we have that $|e\, x| \leqslant u_{j,x} \cdot v_{j,x}$; hence $e\, x\, (u_{j,x} \cdot v_{j,x}) = g\, x$. So for all $j : \mathtt{R}$ with $\mathtt{0} \leqslant j \leqslant \mathtt{1}$ we have a path $\langle i \leqslant u_{j,x}\rangle (e\, x\, (i \cdot v_{j,x})) : f\, x \sim g\, x$ in $A\, x$. When $j = \mathtt{0}$ this path is $\mathtt{happly}(\mathtt{funext}\, e)\, x$ (because $u_{\mathtt{0},x} = \mathtt{1}$ and $v_{\mathtt{0},x} = |e\, x|$); when $j = \mathtt{1}$ the path is $e\, x$ (because $u_{\mathtt{1},x} = |e\, x|$ and $v_{\mathtt{1},x} = \mathtt{1}$). Therefore we can define

$$\varepsilon\, e :\equiv \langle j \leqslant \mathtt{1}\rangle \lambda x \to \langle i \leqslant u_{j,x}\rangle (e\, x\, (i \cdot v_{j,x})) \tag{30}$$

to get the desired path $\mathtt{happly}(\mathtt{funext}\, e) \sim e$ in $\prod_{x:\Gamma}(f\, x \sim g\, x)$. Since for any $p : f \sim g$ it is the case that $p = \langle i \leqslant |p|\rangle (p\, i)$ and $\mathtt{funext}(\mathtt{happly}\, p) = \langle i \leqslant \mathtt{1}\rangle (p\, (i \cdot |p|))$, a similar argument to the one for ε shows that

$$\eta\, p :\equiv \langle j \leqslant \mathtt{1}\rangle \langle i \leqslant (\mathtt{1} - j) \cdot |p| + j\rangle (p\, (i \cdot (\mathtt{1} - j + j \cdot |p|))) \tag{31}$$

gives a path $p \sim \mathtt{funext}(\mathtt{happly}\, p)$ in $f \sim g$. ◀

5 Weak Fibrations

We leave to a sequel the study of universes and univalence [30, section 2.10] in this setting. However, initial investigation suggests that a weaker notion of fibration than in Definition 3.1 may be needed for this. To see why, we show that an essential aspect of univalence already holds for order-ringed toposes $(\mathcal{E}, \mathtt{R})$: *given a contractible object $C \in \mathcal{E}$, there is a family $\hat{C} \in \mathcal{E}(\mathtt{R_+})$ that gives a Moore path of types of shape $\mathtt{1}$ with $\hat{C}\, \mathtt{0} \cong C$ and $\hat{C}\, \mathtt{1} \cong 1$.*

Recall that C is *contractible* [30, section 3.11] if there is some $c_0 : C$ (the centre of contraction) and a function $\kappa : \prod_{x:C}(x \sim c_0)$. Given such a C, one way to get $\hat{C} \in \mathcal{E}(\mathtt{R_+})$ as above is to construct a cone over C and take the family of slices through the cone starting at the base and moving to its vertex. Specifically, for each $i : \mathtt{R_+}$ we can define

$$\hat{C}\, i \ :\equiv\ C/{\approx_i} \quad \text{where } \approx_i \text{ is the equivalence relation: } x \approx_i y \ :\equiv\ (x = y \ \vee\ \mathtt{1} \leqslant i)\,. \tag{32}$$

Write $[x]_i$ for the $\approx_i$-equivalence class of $x : C$. If $i \geqslant \mathtt{1}$, then $\approx_i$ is the indiscrete equivalence relation identifying all $x : C$ with c_0 and so $\hat{C}\, i = \{[c_0]_i\} \cong 1$. When $i = \mathtt{0}$, so long as $\mathtt{R}$ is *non-trivial*

$$\neg(\mathtt{0} = \mathtt{1}) \tag{33}$$

then $\approx_{\mathtt{0}}$ is the discrete equivalence relation $(x \approx_{\mathtt{0}} y \Leftrightarrow x = y)$ and hence $\hat{C}\, \mathtt{0} = C/{\approx_{\mathtt{0}}} \cong C$.

If there were a universe for small fibrations (with respect to some given notion of smallness), then for the family $(\hat{C}\, i \mid i : \mathtt{R_+})$ to determine a path in it, in addition to C being small, we would also need a tap structure for $\hat{C}$ (Definition 3.1). It is not clear why such a structure should exist. However, $\hat{C}$ is an instance of the following weaker notion of fibration where the equality (14) is replaced by a path:

▶ **Definition 5.1** (Weak fibration)**.** Given an object $\Gamma \in \mathcal{E}$, a *weak fibration* over Γ is specified by a family $A \in \mathcal{E}(\Gamma)$ together with functions $\alpha : \prod_{x,y:\Gamma}(x \sim y) \to (A\,x \to A\,y)$ and $\alpha' : \prod_{x:\Gamma}\prod_{a:A\,x}(a \sim \alpha\,x\,x\,(\mathtt{idp}\,x)\,a)$. We write $\mathbf{wFib}(\Gamma)$ for the set of weak fibrations over Γ.

Using the contraction structure c_0, κ of C, we can get such a weak fibration over $\Gamma = \mathtt{R}_+$ with $A = \hat{C}$ and the functions α and α' well-defined by

$$\begin{aligned} \alpha\,i\,j\,p\,[x]_i &:\equiv [\kappa\,x\,(i \cdot \mid \kappa\,x\,\mid)]_j && (i, j : \mathtt{R}_+, p : i \sim j, x : C) \\ \alpha' i\,[x]_i &:\equiv \langle j \leqslant i\rangle [\kappa\,x\,(j \cdot \mid \kappa\,x\,\mid)]_i && (i : \mathtt{R}_+, x : C) \end{aligned}$$

Restricting this to the end point $i = \mathtt{0}$ gives the identity transport function, which indeed corresponds under the isomorphism $\hat{C}\,\mathtt{0} \cong C$ to that for C regarded as a (weakly) fibrant object. More generally, given $\Gamma \in \mathcal{E}$, if we start with a family $C \in \mathbf{wFib}(\Gamma)$ all of whose fibres are contractible, the method outlined above constructs $\hat{C} \in \mathbf{wFib}(\Gamma \times \mathtt{R}_+)$ with $\hat{C}(\_, \mathtt{0})$ and C isomorphic weak fibrations over Γ and with $\hat{C}(\_, \mathtt{1})$ isomorphic to the terminal weak fibration over Γ.

Even though Definition 5.1 might seem to be a very weak notion of fibration, we have found that versions of Theorems 3.4 and 3.5 hold for it, albeit with more complicated proofs. The same is true for Theorem 3.9 except that one gets only *propositional* identity types, where there is a path between $(\mathtt{J}_A\,B\,\beta)\,\mathtt{Refl}_A$ and β, rather than an equality in $\mathcal{E}$ (cf. [6, section 9.1] and [31]).

6 A Gros Topos Model

In the previous sections we have seen how any order-ringed topos $(\mathcal{E}, \mathtt{R})$ (Definition 4.1) gives rise to a model of Martin-Löf type theory with intensional identity types given by Moore paths on $\mathtt{R}$. If $\mathtt{R}$ is trivial, that is, satisfies $\mathtt{0} = \mathtt{1}$, then existence of a Moore path just coincides with extensional equality in the topos $\mathcal{E}$. If the object $\mathtt{R}$ is non-trivial, but has decidable equality in $\mathcal{E}$ (for example, if it is the object of integers, or of rationals), then there is a path $\langle i \leqslant \mathtt{1}\rangle(\mathtt{if}\ i = \mathtt{0}\ \mathtt{then}\ \mathtt{true}\ \mathtt{else}\ \mathtt{false}) : \mathtt{true} \sim \mathtt{false}$ in the object of Booleans and so in this case the model of type theory we get is logically degenerate. Therefore, when searching for examples of order-ringed toposes, we should at least look for ones with $\mathtt{R}$ non-trivial and not decidable. In fact we will show that there are order-ringed toposes for which the associated model of type theory has identity types that are not necessarily truncated at any level. In other words, ones in which iterated identity types $\mathtt{Id}_A, \mathtt{Id}_{\mathtt{Id}_A}, \mathtt{Id}_{\mathtt{Id}_{\mathtt{Id}_A}}, \ldots$ can be homotopically non-trivial for any level of iteration. An interesting way of demonstrating that is by giving an $(\mathcal{E}, \mathtt{R})$ for which the homotopy types of a rich collection of topological spaces (including all the n-dimensional spheres, say) are faithfully represented by the internal, $\mathtt{R}$-based notion of homotopy on a corresponding collection of objects of the topos $\mathcal{E}$. We give one such order-ringed topos in this section.

The topos

We use a topos of sheaves $\mathbf{Sh}(\mathbf{T}, J)$ [19, III] for the following site $(\mathbf{T}, J)$. Let **Haus** denote the category of Hausdorff topological spaces and continuous functions. We take the small category $\mathbf{T}$ to be the least full subcategory of **Haus** containing the reals $\mathbb{R}$ and closed under the following operations:

(a) If $X, Y \in \mathbf{T}$, then $\mathbf{T}$ contains the product space $X \times Y$.
(b) If $X \in \mathbf{T}$ and $C \subseteq X$ is a closed subset, then C (with the subspace topology) is in $\mathbf{T}$.
(c) If $X, Y \in \mathbf{T}$ and X is locally compact, then $\mathbf{T}$ contains the exponential space Y^X (the set of continuous functions from X to Y endowed with the compact-open topology).

Since the spaces in $\mathbf{T}$ are Hausdorff, equalizers of continuous functions give closed subspaces and hence by (a) and (b) we have that $\mathbf{T}$ is closed under taking finite limits in **Haus**. Hence $\mathbf{T}$ contains $\mathbb{R}_+$ (as well as $[0,1]$ and all the spheres S^n for any $n \in \mathbb{N}$). Then by (c) we have that $\mathbf{T}$ is closed under taking Moore path spaces (with their usual topology):

$$X \in \mathbf{T} \;\Rightarrow\; MX :\equiv \{(f,r) \in X^{\mathbb{R}_+} \times \mathbb{R}_+ \mid (\forall r' \geq r)\; f\, r' = f\, r\} \in \mathbf{T}\,. \tag{34}$$

We define a coverage [14, Definition A2.1.9] on $\mathbf{T}$ as follows. Following Dyer and Eilenberg [9] we say that a set S of closed subsets of a topological space X is a *local cover* if for each $x \in X$ there is a finite subset $\{C_1, \ldots, C_n\} \subseteq S$ with x in the interior of $C_1 \cup \cdots \cup C_n$. Of course every finite cover of X by closed subsets is trivially a local cover. Note that if $f : Y \to X$ in $\mathbf{T}$ and S is a local cover of X, then $f^{-1}S :\equiv \{f^{-1}C \mid C \in S\}$ is a local cover of Y. Hence we get a coverage on $\mathbf{T}$ with $J(X) :\equiv \{S \mid S \text{ is a local cover of } X\}$ for each $X \in \mathbf{T}$.

Given a functor $\Gamma : \mathbf{T}^{\mathrm{op}} \to \mathbf{Set}$, if $C \subseteq X$ is a closed subspace of a space in $\mathbf{T}$, then for each $x \in \Gamma\, X$ we will just write $x|_C$ for the element $\Gamma\, i\, x \in \Gamma\, C$, where the $\mathbf{T}$-morphism $i : C \to X$ is the inclusion function. Recall that Γ is a sheaf with respect to J iff for all $X \in \mathbf{T}$, $S \in J(X)$ and all S-indexed families $(x_C \in \Gamma\, C \mid C \in S)$ satisfying $(\forall C, D \in S)\; x_C|_{C\cap D} = x_D|_{C\cap D}$, there is a unique $x \in \Gamma\, X$ with $(\forall C \in S)\; x_C = x|_C$. The topos $\mathbf{Sh}(\mathbf{T}, J)$ is the full subcategory of the functor category $[\mathbf{T}^{\mathrm{op}}, \mathbf{Set}]$ whose objects are sheaves.

The ordered commutative ring object

Let $\mathrm{Y} : \mathbf{T} \to [\mathbf{T}^{\mathrm{op}}, \mathbf{Set}]$ denote the Yoneda embedding for the small category $\mathbf{T}$. Because elements of $J(X)$ are local covers of $X \in \mathbf{T}$, for each $S \in J(X)$ if we have a family of continuous functions $f_C : C \to X'$ for each $C \in S$ that agree where they overlap ($f_C|_{C\cap D} = f_D|_{C\cap D}$), then the unique function $f : X \to X'$ that agrees with each of them ($f_C = f|_C$) is necessarily continuous. Hence each representable presheaf $\mathrm{Y}\, X' = \mathbf{T}(\_, X')$ is a sheaf (in other words the coverage J is sub-canonical) and the Yoneda embedding gives a functor $\mathrm{Y} : \mathbf{T} \to \mathbf{Sh}(\mathbf{T}, J)$.

The axioms for *partially* ordered commutative rings make sense in any category with finite limits once we have an interpretation of the binary operation $\leqslant$, the constants $\mathtt{0}, \mathtt{1}$ and the operations $+, \text{-}, \cdot$; and satisfaction of those axioms is preserved by functors that preserve finite limits. Since $\{(x,y) \in \mathbb{R} \times \mathbb{R} \mid x \leq y\}$ is a closed subset of $\mathbb{R} \times \mathbb{R}$ and the usual ring operations on $\mathbb{R}$ are continuous, it follows that $\mathbb{R}$ is a partially ordered commutative ring object in the finitely complete category $\mathbf{T}$. Then since Y preserves finite limits, the representable sheaf $\mathtt{R} :\equiv \mathrm{Y}\mathbb{R}$ is a partially ordered commutative ring object in the topos $\mathbf{Sh}(\mathbf{T}, J)$. In fact it is totally ordered, that is, satisfies (2). This is simply because we have a local cover $\{\{(x,y) \mid x \leq y\}, \{(x,y) \mid y \leq x\}\} \in J(\mathbb{R} \times \mathbb{R})$.[2]

Homotopy types in $\mathbf{Sh}(\mathbf{T}, J)$

As in section 3 we get a CwF **Fib** based on $\mathbf{Sh}(\mathbf{T}, J)$ with intensional identity types given by Moore paths from $\mathtt{R} = \mathrm{Y}\mathbb{R}$. Consider the congruence on the category $\mathbf{Sh}(\mathbf{T}, J)$ that identifies morphisms $\gamma, \delta : \Delta \to \Gamma$ when there is a global section of $\mathtt{Id}_{\Delta\to\Gamma}\, \gamma\, \delta$. Let $\mathbf{Ho}(\mathbf{Sh}(\mathbf{T}, J))$ denote the associated quotient category.

[2] This fact is the motivation for considering a site based on covers by closed subsets rather than the more familiar case of open covers used for Giraud's gros topos [10, IV, 2.5]. However, as Spitters has pointed out [private communication], in view of [13, Theorem 8.1] we could have used the reals in Johnstone's *topological topos* instead of $\mathbf{Sh}(\mathbf{T}, J)$ in this section.

By Theorem 4.2, two morphisms $\gamma, \delta : \Delta \to \Gamma$ are identified in $\mathbf{Ho}(\mathbf{Sh}(\mathbf{T}, J))$ iff $\prod_{x:\Delta}(\gamma\, x \sim \delta\, x)$ has a global section. This is equivalent to requiring the existence of a morphism $H : \Delta \to \wp\Gamma$ in $\mathbf{Sh}(\mathbf{T}, J)$ satisfying $\partial^- \circ H = \gamma$ and $\partial^+ \circ H = \delta$, where

$$\begin{aligned} &\wp\Gamma :\equiv \{(f, i) : (\mathtt{R}_+ \to \Gamma) \times \mathtt{R}_+ \mid (\forall j \geqslant i)\ f\, j = f\, i\} \\ &\partial^-, \partial^+ : \wp\Gamma \to \Gamma \qquad \partial^-(f, i) :\equiv f\, 0 \qquad \partial^+(f, i) :\equiv f\, i \end{aligned} \tag{35}$$

is the total object of the family of Moore path objects (6) with associated source and target morphisms. Note that since $\mathtt{R} = \mathrm{Y}\mathbb{R}$, it follows that we also have $\mathtt{R}_+ \cong \mathrm{Y}(\mathbb{R}_+)$. Furthermore, as well as preserving finite limits the Yoneda embedding preserves any exponentials that happen to exist. It follows that for each representable sheaf $\mathrm{Y}X$ its total path object is representable by the Moore path space of X (34): $\wp(\mathrm{Y}X) \cong \mathrm{Y}(MX)$; and under this isomorphism the source and target morphisms ∂^-, ∂^+ correspond to the representable morphisms induced by the usual source and target functions for MX. Since Y is full and faithful, it follows that for two continuous functions $f, g : X \to X'$ in $\mathbf{T}$, the morphisms $\mathrm{Y}f$ and $\mathrm{Y}g$ get identified in $\mathbf{Ho}(\mathbf{Sh}(\mathbf{T}, J))$ iff f and g are homotopic in the classical sense. Thus Y induces a full and faithful embedding of the category $\mathbf{Ho}(\mathbf{T})$ of homotopy types of spaces in $\mathbf{T}$ into $\mathbf{Ho}(\mathbf{Sh}(\mathbf{T}, J))$. Since $\mathbf{T}$ contains all the spheres S^n, we deduce that identity types in this particular **Fib** are not necessarily truncated at any level of iteration.

7 Related Work

The classical topological notion of Moore path is a standard, if somewhat niche topic within homotopy theory. The Schedule Theorem of Dyer and Eilenberg [9] for globalising Hurewicz fibrations is a nice example of their usefulness. They have been used in connection with higher-dimensional category theory by Kapranov and Voevodsky [16] and by Brown [5].

Although our use of constructive algebra within toposes to make models of intensional type theory appears to be new, we are not the only ones to consider using some form of path with strictly unitary and associative composition to model identity types with a judgemental computation rule. Van den Berg and Garner [32] use topological Moore paths and a simplicial version of them to get instances of their notion of *path object category* for modelling identity types. The results of section 2 show that any ordered abelian group in a topos induces a path object category structure on that topos; and since the notion of fibration we use in section 3 in fact coincides with the one used in [32] (see Proposition 6.1.5 of that paper), one can get alternative, more abstract proofs of Theorems 3.4 and 3.9 from the results of Van den Berg and Garner. However, the concrete calculations in the internal language that we give are quite simple by comparison; and this approach proves its worth in section 4, whose results are new as far as we know.

The forthcoming PhD thesis of North [25] uses a category-theoretic abstraction of the notion of Moore paths, called *Moore relation systems*, as part of a complete analysis of when a weak factorization system gives a model (in terms of display map categories, rather than CwFs) of identity-, Σ- and Π-types. A Moore relation system is a piece of category theory comparable to our use of ordered abelian groups in categorical logic in section 2; it would be interesting to see if it can be extended in the way we extended from groups to rings in section 4 in order to validate function extensionality.

Spitters [27, section 3] attempts to use a somewhat different formulation of Moore path in the cubical topos [6]. His notion is the reflexive-transitive closure of the usual path types given by the bounded interval. For better properties and to get a closer relationship with our version, one would like to quotient these cubical "Spitters-Moore" path objects up to

degenerate paths; but the undecidability of degeneracy seems to stop one being able to do that while retaining the (uniform) Kan-fibrancy of such path objects. Here we can side-step such issues, since notably our models manage to avoid using a notion of Kan fibrancy at all.

8 Future Work

There are two directions in which we would like to develop this work. First, the calculations about type structure in this paper using constructive algebraic Moore paths, particularly those using the bounded abstraction notation (Definition 2.4), are strikingly simple compared with ones based upon Kan filling [17, 3, 6]. It would be interesting and possibly useful to embody them in a constructive type theory with Moore path types and in which any type families describable in the theory are automatically fibrant. The inspiration for this is the Cubical Type Theory of Cohen, Coquand, Huber and Mörtberg [6].

Secondly, are there order-ringed toposes for which the associated CwF **Fib** constructed as in section 3 contains universes that satisfy Voevodsky's *univalence axiom* [30, section 2.10]? Our initial analysis of what is needed to satisfy univalence (see section 5) suggests that one just needs an order-ringed topos containing a fibration that is weakly generic for (weak) fibrations whose fibres are small with respect to some given external Grothendieck universe. Such universes in the classical simplicial sets [18] and constructive cubical sets [6] models of type theory make use of a modified form of the Hofmann-Streicher [12] construction in *presheaf* categories. To be sure, there are order-ringed toposes $(\mathcal{E}, \mathtt{R})$ for which $\mathcal{E}$ is a presheaf topos (with $\mathtt{R}$ non-trivial and not decidable). For example one can take $\mathcal{E} = [\mathbf{C}, \mathbf{Set}]$ where $\mathbf{C}$ is a suitable small full subcategory of the category of ordered commutative rings in $\mathbf{Set}$ and where $\mathtt{R}$ is the forgetful functor $\mathbf{C} \to \mathbf{Set}$. However, this $\mathtt{R}$ is not representable and $\mathbf{C}^{\mathrm{op}}$ is not cartesian; and that seems to block the construction of a (weak) tap structure for the generic family over the Hofmann-Streicher universe of small fibrations on representables. Given the crucial, disjunctive axiom (2) upon which everything in this paper depends, one is led to consider sheaf toposes instead of presheaf toposes, as we did in section 6. Streicher [29] points out that the basic Hofmann-Streicher universe construction works for sheaf toposes through a suitable use of sheafification. However, as yet we have not been able to extend that to work for the sheafification of the presheaf universe of small families *with tap structure* over representables, for the site in section 6. Although that presheaf universe is not a sheaf, it is probably a *stack of groupoids* [28, Tag 02ZH]. (To see this one would have to prove a version of globalisation [9] for fibrations in $\mathbf{Sh}(\mathbf{T}, J)$.) In which case it might be fruitful to consider stack-based models of univalent type theory of the kind considered by Coquand, Mannaa and Ruch [7], or higher-dimensional versions thereof.

Acknowledgements. We are very grateful to Benno van den Berg and Bas Spitters for discussions about the material in this paper.

References

1 Agda. URL: `http://wiki.portal.chalmers.se/agda/pmwiki.php`.

2 S. Awodey. Natural models of homotopy type theory. *Mathematical Structures in Computer Science*, [tba]:1–46, 2016.

3 M. Bezem, T. Coquand, and S. Huber. A model of type theory in cubical sets. In *Proc. TYPES 2013*, volume 26 of *LIPIcs*, pages 107–128, 2014.

4 N. Bourbaki. *Algèbre*, volume II of *Eléments de mathématiques*. Masson, 1981.

5 R. Brown. Moore hyperrectangles on a space form a strict cubical omega-category. *ArXiv e-prints*, arXiv:0909.2212v4 [math.CT], 2009.

6 C. Cohen, T. Coquand, S. Huber, and A. Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. *ArXiv e-prints*, arXiv:1611.02108 [cs.LO], 2016.

7 T. Coquand, B. Mannaa, and F. Ruch. Stack semantics of type theory. *ArXiv e-prints*, arXiv:1701.02571v1 [cs.LO], 2017.

8 P. Dybjer. Internal type theory. In S. Berardi and M. Coppo, editors, *Types for Proofs and Programs*, volume 1158 of *Lecture Notes in Computer Science*, pages 120–134. Springer Berlin Heidelberg, 1996.

9 E. Dyer and S. Eilenberg. Globalizing fibrations by schedules. *Fundamenta Mathematicae*, 130(2):125–136, 1988.

10 A. Grothendieck and J. L. Verdier. *Théorie des topos (SGA 4, exposés I–VI)*, volume 269-270 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1972.

11 M. Hofmann. Syntax and semantics of dependent types. In A. M. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, pages 79–130. CUP, 1997.

12 M. Hofmann and T. Streicher. Lifting Grothendieck universes. Unpublished note, 1999.

13 P. T. Johnstone. On a topological topos. *Proceedings of the London Mathematical Society*, s3-38(2):237–271, 1979.

14 P. T. Johnstone. *Sketches of an Elephant, A Topos Theory Compendium, Volumes 1 and 2*. Number 43–44 in Oxford Logic Guides. Oxford University Press, 2002.

15 D. M. Kan. A combinatorial definition of homotopy groups. *Annals of Mathematics*, 67(2):282–312, 1958.

16 M. M. Kapranov and V. A. Voevodsky. ∞-groupoids and homotopy types. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 32(1):29–46, 1991.

17 C. Kapulkin and P. L. Lumsdaine. The simplicial model of univalent foundations (after Voedodsky). *ArXiv e-prints*, arXiv:1211.2851v4 [math.LO], 2016.

18 P. L. Lumsdaine and M. A. Warren. The local universes model: an overlooked coherence construction for dependent type theories. *ACM Trans. Comput. Logic*, 16(3):23:1–23:31, 2015.

19 S. MacLane and I. Moerdijk. *Sheaves in Geometry and Logic. A First Introduction to Topos Theory*. Springer, 1992.

20 M. E. Maietti. Modular correspondence between dependent type theories and categories including pretopoi and topoi. *Math. Structures in Comp. Science*, 15:1089–1149, 2005.

21 P. Martin-Löf. An intuitionistic theory of types: Predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium 1973*, pages 73–118. North-Holland, 1975.

22 P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, Napoli, 1984.

23 I. Moerdijk and E. Palmgren. Wellfounded trees in categories. *Annals of Pure and Applied Logic*, 104(1):189–218, 2000.

24 J. C. Moore. Le théorème de Freudenthal, la suite exacte de James et l'invariant de Hopf généralisé. *Séminaire Henri Cartan*, 7(2):exp. 22, 1–15, 1954-1955.

25 P. R. North. *Type Theoretic Weak Factorization Systems*. PhD thesis, University of Cambridge, 2017.

26 I. Orton and A. M. Pitts. Axioms for modelling cubical type theory in a topos. In *Proc. CSL 2016*, volume 62 of *LIPIcs*, pages 24:1–24:19, 2016.

27 B. Spitters. Cubical sets and the topological topos. *ArXiv e-prints*, arXiv:1610.05270v1 [cs.LO], 2016.

28 The Stacks Project Authors. Stacks project. `http://stacks.math.columbia.edu`, 2017.

29 T. Streicher. Universes in toposes. In L. Crosilla and P. Schuster, editors, *From Sets and Types to Topology and Analysis, Towards Practicable Foundations for Constructive Mathematics*, volume 48, chapter 4, pages 78–90. OUP, 2005.

30 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations for Mathematics*. Institute for Advanced Study, 2013.

31 B. van den Berg. Path categories and propositional identity types. *ArXiv e-prints*, arXiv:1604.06001 [math.CT], 2016.

32 B. van den Berg and R. Garner. Topological and simplicial models of identity types. *ACM Trans. Comput. Logic*, 13(1):3:1–3:44, 2012.

On Dinaturality, Typability and $\beta\eta$-Stable Models

Paolo Pistone

Dipartimento di Matematica e Fisica, Università Roma Tre, Rome, Italy
paolo.pistone@uniroma3.it

Abstract

We present two results which relate dinaturality with a syntactic property (typability) and a semantic one (interpretability by $\beta\eta$-stable sets). First, we prove that closed dinatural λ-terms are simply typable, that is, the converse of the well-known fact that simply typable closed terms are dinatural. The argument exposes a syntactical aspect of dinaturality, as λ-terms are type-checked by computing their associated dinaturality equation. Second, we prove that a closed λ-term belonging to all $\beta\eta$-stable interpretations of a simple type must be dinatural, that is, we prove dinaturality by semantical means. To do this, we show that such terms satisfy a suitable version of binary parametricity and we derive dinaturality from it.

By combining the two results we obtain a new proof of the completeness of the $\beta\eta$-stable semantics with respect to simple types. While the completeness of this semantics is well-known in the literature, the technique here developed suggests that dinaturality might be exploited to prove completeness also for other, less manageable, semantics (like saturated families or reducibility candidates) for which a direct argument is not yet known.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Dinaturality, simply-typed lambda-calculus, $\beta\eta$-stable semantics, completeness

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.29

1 Introduction

Dinaturality is a property often considered in investigations on polymorphism and can be described in at least three ways. Firstly, as a categorial property, that is, as the fact that a program of type $\sigma \to \tau$ (or proofs of an implication) corresponds to a *dinatural transformation* between the multivariant[1] functors F_σ, F_τ associated to the types σ and τ. Secondly, it can be described as a purely syntactic relation between a λ-term and its type: since the functorial action $F_\sigma[f, B], F_\sigma[A, f]$ of a type σ can be expressed by simply typable λ-terms H_σ, K_σ, the dinaturality of a term M with respect to σ can be expressed by an equation relating M, H_σ and K_σ. Thirdly, it can be presented as a special case of Reynolds' relational parametricity ([10]), namely the case in which only functional relations are considered (see [9]).

It is a well-known fact that simply-typed λ-terms are dinatural in all three senses just mentioned (see [1, 5, 9]). In this paper we establish two facts about dinaturality which put this notion at the center of a chain of arrows from a semantic notion (interpretability by $\beta\eta$-stable sets) and a syntactic notion (simple typability):

Dinaturality $\Rightarrow$ Typability. We prove that the converse of the aforementioned fact that simply-typable closed λ-terms are dinatural holds: if a closed $\beta\eta$-normal λ-term is dinatural (in the syntactical sense), then it can be simply typed (Theorem 14). It is

[1] that is, functors whose arguments can be divided into a covariant and a contravariant part, corresponding to variables occurring positively and negatively in the type.

shown that the dinaturality condition can be exploited to recursively type-check λ-terms: a typing derivation for a λ-term is constructed from the computation of the equation expressing its dinaturality.

Interpretability $\Rightarrow$ Dinaturality. The semantics of $\beta\eta$-stable sets of λ-terms was introduced by Krivine as a semantics of second order functional arithmetic ([7]). We prove that an adapted version of binary parametricity (Definition 21) can be expressed in this semantics and we establish that, for all simple type σ, a closed λ-term belonging to all interpretations of σ must be parametric in σ (Theorem 24). Syntactic dinaturality is then shown to follow from parametricity with respect to a particular relation assignment (Theorem 27).

The parametricity and dinaturality of a λ-term are usually established by induction over a typing derivation of the term; on the contrary, parametricity (and *a fortiori* dinaturality, Theorem 27) is here established for a class of untyped λ-terms (those which are interpretable in the $\beta\eta$-stable semantics) by induction over simple types (Theorem 24). Moreover, this last result can be thought of as a first step towards the investigation of parametricity and dinaturality in the context of the reducibility semantics of typed λ-calculi (like the different saturated families considered in [7] or Girard's reducibility candidates [4]). These semantics are commonly used to prove normalization results but their relationship with parametricity and dinaturality, as well as the completeness issue, are still unclear.

By composing the two arrows above we obtain a new argument for the completeness of the $\beta\eta$-stable semantics: for every simple type σ, if a closed λ-term belongs to $|\sigma|_{\mathcal{M}}$ for all interpretation $\mathcal{M}$, then it is $\beta\eta$-equivalent to a term having type σ. This result had already been established by a direct argument (see [6, 8]) and extended to so-called *positive second order types* in Krivine's system $\mathcal{AF}$ (see [3]). However, the arguments developed in these papers cannot be generalized in a straightforward way to the reducibility semantics mentioned above. It might be interesting to see if a semantic proof of parametricity (and, as a consequence, of completeness) can be reproduced in such frames.

The paper is organized as follows: in Section 2 we introduce a tree characterization of simple typability, which will be exploited in the proof of Theorem 14; in Section 3 we recall syntactic dinaturality and in Section 4 we prove Theorem 14. In Section 5 we recall the $\beta\eta$-stable semantics and we define a variant of parametricity based on $\beta\eta$-stable relations; in Section 6 we prove the parametricity Theorem 24 and in Section 7 we apply it to obtain dinaturality (Theorem 27); finally, in Section 8 we briefly discuss some issues related to completeness and other reducibility semantics.

2 Tree representation of simple typability

In this section we present a characterization of typability for closed $\beta\eta$-normal λ-terms based on a confrontation between the tree of the term $\mathcal{T}(M)$ and the tree of its assigned type $\mathcal{T}(\sigma)$.

In the following, by the letters $M, N, P, \dots$ we will indicate elements of the set Λ of untyped λ-terms, subject to usual α-equivalence. By the letters $x, y, z, \dots$ we will indicate λ-*variables*, i.e. the variables that might occur free or bound in untyped λ-terms. By the expression $M_1M_2M_3 \dots M_n$ we will indicate the term $(\dots((M_1M_2)M_3)\dots M_n)$.

As usual, the relation $M \to_\beta^* N$ indicates the existence of a sequence of β-reduction steps from M to N and the relation $M \simeq_\beta N$ (resp. $M \simeq_{\beta\eta} N$) indicates that M and N are β-equivalent (resp. $\beta\eta$-equivalent).

▶ **Definition 1** (The simply typed λ-calculus $\lambda_\to$ "à la Curry")**.** Given a countable set $\mathcal{V} = \{Z_1, Z_2, Z_3, \dots, \}$ of symbols, called *type variables* (or, simply, variables when no ambiguity

occurs), the set $\mathcal{T}$ of simple types is defined inductively as follows:

$$Z_u \in \mathcal{V} \;\Rightarrow\; Z_u \in \mathcal{T} \tag{1}$$
$$\sigma, \tau \in \mathcal{T} \;\Rightarrow\; \sigma \to \tau \in \mathcal{T} \tag{2}$$

A *type declaration* is an expression of the form $x : \sigma$, where x is a term variable and σ is a type. A *context* Γ is a finite set of type declarations (where no two distinct declarations $x : \sigma, x : \tau$, with $\sigma \neq \tau$ appear). A *judgement* is an expression of the form $\Gamma \vdash M : \sigma$, where Γ is a context, M a term and σ a type.

The typing derivations of $\lambda_{\to}$ are generated by the following rules:

$$\boxed{\frac{}{\Gamma \cup \{x : \sigma\} \vdash x : \sigma}\ (id) \qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}\ (\to E) \qquad \frac{\Gamma \cup \{x : \sigma\} \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\ (\to I)} \tag{3}$$

The *arity* $ar(M)$ of a λ-term is the positive integer n such that M can be written as $\lambda x_1 . \ldots . \lambda x_n . M'$ with M' either a variable or an application. The *arity* $ar(\sigma)$ of a simple type $\sigma \in \mathcal{T}_0$ is the positive integer n such that $\sigma = \sigma_1 \to \cdots \to \sigma_n \to Z_u$. We indicate by $FV(M)$ (resp. $BV(M)$) the set of free (resp. bound) variables of M.

▶ **Definition 2.** A *path* $\pi = p_1 \ldots p_k$ is a finite sequence of non zero positive integers. We denote ϵ the empty path and $\mathbb{N}^*$ the set of all paths. If π is the path $p_1 \ldots p_k$, then for every $p \in \mathbb{N}$, $\pi * p$ denotes the path $p_1 \ldots p_k p$ and $p * \pi$ the path $p p_1 \ldots p_n$. The *length* of a path $\ell(\pi)$ is the length of the sequence π. A partial order $\leq$ over paths is defined by letting $\pi \leq \pi'$ if $\pi = p_1 \ldots p_k$ and $\pi' = p_1 \ldots p_k p_{k+1} \ldots p_{k'}$ for some $k, k' \geq 0$.

A *tree* is a set $T \subseteq \mathbb{N}^*$ containing ϵ and such that, if $\pi \in T$ and $\pi' \leq \pi$, then $\pi' \in T$. If $T_1, \ldots, T_n$ are trees, for some $n \geq 1$, then $(T_1, \ldots, T_n)$ is the tree consisting of the empty sequence and all sequences of the form $i * \pi$, for $\pi \in T_i$ and $i \leq n$.

For σ and M, respectively, a simple type and a closed $\beta\eta$-normal λ-term, we define the trees $\mathcal{T}(\sigma)$ and $\mathcal{T}(M)$:

▶ **Definition 3** (Tree of a type)**.** Let σ be a simple type. We associate with σ a tree $\mathcal{T}(\sigma)$ defined by induction as follows:

- if $\sigma = Z_u$, then $\mathcal{T}(\sigma) = \{\epsilon\}$;
- if $\sigma = \sigma_1 \to \cdots \to \sigma_n \to Z_u$, then $\mathcal{T}(\sigma) = \big(\mathcal{T}(\sigma_1), \ldots, \mathcal{T}(\sigma_n), \mathcal{T}(Z_u)\big)$

Conversely, with every $\pi \in \mathcal{T}(\sigma)$ we can associate a unique subtype σ_π of σ:

- $\sigma_\epsilon := \sigma$;
- $\sigma_{\pi * i} := \tau_i$, for $1 \leq i \leq ar(\sigma_\pi) + 1$, where $\sigma_\pi = \tau_1 \to \cdots \to \tau_{ar(\sigma_\pi)} \to Z_u$ where we put $\tau_{ar(\sigma_\pi)+1} = Z_u$.

▶ **Definition 4** (Tree of a closed $\beta\eta$-normal λ-term)**.** Let M be a closed $\beta\eta$-normal λ-term. We associate with M a tree $\mathcal{T}(M)$ defined by induction on the number of applications of M as follows:

- if $M = \lambda x_1 . \ldots . \lambda x_h . x_i$, then $\mathcal{T}(M) = \epsilon$;
- if $M = \lambda x_1 . \ldots . \lambda x_h . x_i M_1 \ldots M_p$, for some $h \geq 0$, $1 \leq i \leq h$ and $p \geq 1$, then $\mathcal{T}(M) = \big(\mathcal{T}(\overline{M_1}), \ldots, \mathcal{T}(\overline{M_p})\big)$, where $\overline{M_j} = \lambda x_1 . \ldots . \lambda x_h . M_j$, for $1 \leq j \leq p$.

Conversely, with every $\pi \in \mathcal{T}(M)$ we can associate a unique subterm M_π

- $M_\epsilon := M$;
- $M_{\pi * j} := N_j$, for $1 \leq j \leq q$, where $M_\pi = \lambda x_1 . \ldots . \lambda x_h . y N_1 \ldots N_q$.

For $\pi \in \mathcal{T}(M)$, we let $p_M(\pi)$ be the minimum positive integer $p \geq 0$ such that $\pi * (p+1) \notin \mathcal{T}(M)$.

Let $Var(M) \subset \mathcal{T}(M) \times \mathbb{N}$ indicate the set of all pairs (π, i) such that $1 \leq i \leq ar(M_\pi)$. Any pair $(\pi, i) \in Var(M)$ uniquely determines a variable appearing in M: it is the i-th variable abstracted in M_π.

Given M a closed $\beta\eta$-normal λ-term and σ a simple type, we define two partial maps:

1. a partial map $\mathtt{var} : Var(M) \to \mathcal{T}(\sigma)$ associating variables of M with subtypes of σ;
2. a partial map $\mathtt{subt} : \mathcal{T}(M) \to \mathcal{T}(\sigma)$ associating subterms M_π with subtypes of σ.

The maps $\mathtt{var}$ and $\mathtt{subt}$ approximate the maps (which are always defined for typed λ-terms) associating each variable and each subterm of M with its associated subtype of σ.

Let $h(\pi)$ indicate the pair (π', i) associated with the head variable of M_π. Remark that $\ell(h(\pi)_1) \leq \ell(\pi)$, where $h(\pi)_1$ indicates the first component of the pair $h(\pi)$. The maps $\mathtt{var}$ and $\mathtt{subt}$ are defined as follows:

$$\begin{array}{rcl@{\qquad\qquad}rcl} \mathtt{var}(\epsilon, i) & = & i & \mathtt{subt}(\epsilon) & = & \epsilon \\ \mathtt{var}(\pi * j, i) & = & \mathtt{var}(h(\pi)) * j * i & \mathtt{subt}(\pi * j) & = & \mathtt{var}(h(\pi)) * j \end{array} \tag{4}$$

where it is intended that $\mathtt{var}(\epsilon, i)$ is defined only if $i \leq ar(\sigma)$ and $\mathtt{var}(\pi * j, i)$ is defined only if $\mathtt{var}(h(\pi))$ is defined and if $j \leq ar(\sigma_{\mathtt{var}(h(\pi))})$ and $i \leq ar(\sigma_{\mathtt{var}(h(\pi))*j})$. The definition of $\mathtt{var}(\pi * j, i)$ is a correct recursion since $\ell(h(\pi)_1) < \ell(\pi * j)$. Moreover, $\mathtt{subt}(\pi * j)$ is defined only if $\mathtt{var}(h(\pi))$ is defined and $j \leq ar(\sigma_{\mathtt{var}(h(\pi))})$.

The following theorem shows that typability for a closed $\beta\eta$-normal λ-term M and a simple type σ can be expressed as a relation between $\mathcal{T}(M)$ and $\mathcal{T}(\sigma)$:

▶ **Theorem 5.** *Let M be a $\beta\eta$-normal term, $\sigma_1, \dots, \sigma_n, \tau$ simple types and Γ the context $\{x_1 : \sigma_1, \dots, x_n : \sigma_n\}$. Then $\Gamma \vdash M : \tau$ is derivable if and only if the following hold (where $N = \lambda x_1 \dots . \lambda x_n . M$, $\sigma = \sigma_1 \to \dots \to \sigma_n \to \tau$):*

1. *$\mathtt{var}, \mathtt{subt}$ are defined for all $\pi \in \mathcal{T}(N)$ and $i \geq 1$ such that $(\pi, i) \in Var(N)$;*
2. *$ar(\sigma_{\mathtt{var}(h(\pi))}) = p_N(\pi) + ar(\sigma_{\mathtt{subt}(\pi)}) - ar(N_\pi)$ for all $\pi \in \mathcal{T}(N)$;*
3. *$\sigma_{\mathtt{var}(h(\pi))*(p_N(\pi)+l)} = \sigma_{\mathtt{subt}(\pi)*(ar(N_\pi)+l)}$, for all $1 \leq l \leq ar(\sigma_{\mathtt{subt}(\pi)}) - ar(N_\pi) + 1$.*

Proof. Proof in Appendix A. ◀

From the characterization above we introduce the following notion of *p-fitness*:

▶ **Definition 6.** Let M be a closed $\beta\eta$-normal λ-term and σ a simple type. For $p = 1, 2, 3$ and $\pi \in \mathcal{T}(M)$, we say that M is *p-fit* to σ at π if M satisfies condition $p.$ of Theorem 5 restricted to π. M is *p-fit* if, for all $\pi \in \mathcal{T}(M)$, M is p-fit at π.

3 Syntactic dinaturality

We provide a description of the dinaturality condition for simple types and pure λ-terms. A similar description can be found in [2]. Let $\mathcal{V}_0, \mathcal{V}_1$ be a partition of $\mathcal{V}$ into two disjoint countable sets of type variables $\mathcal{V}_0 = \{X_0, X_1, X_2, \dots\}$ and $\mathcal{V}_1 = \{Y_0, Y_1, Y_2, \dots\}$. For any type σ, whose free variables are $Z_{u_1}, \dots, Z_{u_k} \in \mathcal{V}$, let σ^X_Y (resp. σ^Y_X) denote the result of replacing, in σ, all positive occurrences of Z_{u_l}, for $1 \leq l \leq k$, by the variable X_{u_l} (resp. Y_{u_l}), and all negative occurrences of Z_{u_l} by the variable Y_{u_l} (resp. X_{u_l}). Let then σ_X, σ_Y denote, respectively, σ^X_X and σ^Y_Y.

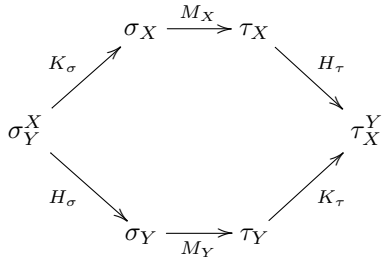
Let, for each $u \geq 0$, f_u be a fresh variable (to which we will assign type $X_u \to Y_u$). For each type σ, we define well-typed terms H_σ, K_σ coding the functorial action of σ: if, for any

type σ, we indicate by $\mathbf{F}_\sigma$ its associated functor, then H_σ codes both morphisms $\mathbf{F}_\sigma[X, f]$ and $\mathbf{F}_\sigma[Y, f]$, as it can be assigned both types $\sigma_X \to \sigma_X^Y$ and $\sigma_Y^X \to \sigma_Y$; similarly, K_σ codes both morphisms $\mathbf{F}_\sigma[f, X]$ and $\mathbf{F}_\sigma[f, Y]$, as it can be assigned both types $\sigma_Y^X \to \sigma_X$ and $\sigma_Y \to \sigma_X^Y$.

The dinaturality condition will be given by the equation below, in which g is a fresh variable (to be declared of type σ_Y^X)

$$H_\tau(M(K_\sigma g)) \simeq_{\beta\eta} K_\tau(M(H_\sigma g)) \tag{5}$$

Equation 5 can be illustrated by the usual hexagonal diagram:



(6)

We define simultaneously the terms H_σ, K_σ by induction over σ:

- if $\sigma = Z_u$, then $H_\sigma := f_u$ and $K_\sigma := \lambda x.x$;
- if $\sigma = \sigma_1 \to \cdots \to \sigma_k \to Z_u$, then

$$H_\sigma := \lambda g.\lambda h_1 \ldots . \lambda h_k . f_u\big(g(K_{\sigma_1} h_1)(K_{\sigma_2} h_2) \ldots (K_{\sigma_k} h_k)\big) \tag{7}$$

and

$$K_\sigma := \lambda g.\lambda h_1 \ldots . \lambda h_k . g(H_{\sigma_1} h_1)(H_{\sigma_2} h_2) \ldots (H_{\sigma_k} h_k) \tag{8}$$

By a simple computation, equation 5 can be rewritten as follows:

$$f_u\big(M(K_{\sigma_1} g_1) \ldots (K_{\sigma_n} g_n)\big) \simeq_{\beta\eta} M(H_{\sigma_1} g_1) \ldots (H_{\sigma_n} g_n) \tag{9}$$

where $\sigma = \sigma_1 \to \cdots \to \sigma_n \to Z_u$ and $g_1, \ldots, g_n$ are fresh variables.

▶ **Proposition 7.** *For all simple type σ, let Γ be the set of all declarations $f_u : X_u \to Y_u$ such that the variable Z_u occurs in σ. Then $\Gamma \vdash H_\sigma : \sigma_X \to \sigma_X^Y$, $\Gamma \vdash H_\sigma : \sigma_Y^X \to \sigma_Y$, $\Gamma \vdash K_\sigma : \sigma_Y^X \to \sigma_X$ and $\Gamma \vdash K_\sigma : \sigma_Y \to \sigma_X^Y$ are derivable.*

We define the set DIN_σ of σ-dinatural terms:

▶ **Definition 8** (dinatural term)**.** Let $\sigma = \sigma_1 \to \cdots \to \sigma_n \to Z_u$ be a simple type and M be a closed λ-term. Then $M \in DIN_\sigma$ if

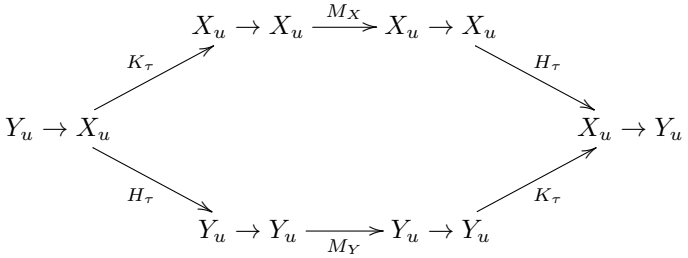
$$f_u\big(M(K_{\sigma_1} g_1) \ldots (K_{\sigma_n} g_n)\big) \simeq_{\beta\eta} M(H_{\sigma_1} g_1) \ldots (H_{\sigma_n} g_n) \tag{10}$$

holds.

▶ **Example 9.** Let σ be the type $(Z_u \to Z_u) \to (Z_u \to Z_u)$. The dinaturality condition associated with a closed λ-term M and σ is the equation

$$f_u\big(M\ \lambda h.g_1(f_u h)\ g_2\big) \simeq_{\beta\eta} M\ \lambda h.f_u(g_1 h)\ f_u g_2 \tag{11}$$

which corresponds to the diagram below (where τ is $Z_u \to Z_u$):



(12)

If M is a closed term of type σ, the $\beta\eta$-normal form of M is of the form $\lambda y.\lambda x.y^n x$, for some $n \geq 0$. Then equation 11 reduces to the true equation

$$f_u\big(\underbrace{g_1(f_u \dots g_1(f_u}_{n \text{ times}} g_2)\dots)\big) \simeq_{\beta\eta} \underbrace{f_u(g_1 \dots f_u(g_1}_{n \text{ times}}(f_u g_2))\dots) \tag{13}$$

showing that $M \in DIN_\sigma$.

The theorem below generalizes this example to an arbitrary closed simply-typed λ-term.

▶ **Theorem 10.** *Let σ be a simple type and M be a closed λ-term. If $\vdash M : \sigma$, then $M \in DIN_\sigma$.*

Proof. Several proofs exist in the literature. [5] prove the fact for the categorial notion of dinaturality, and derive the result for syntactic dinaturality by considering a syntactic category. [2] proves this directly for the syntactic notion of dinaturality. [9] proves dinaturality as a consequence of parametricity. ◀

In the next section we will prove the converse of Theorem 10, showing that dinaturality characterizes closed simply-typable λ-terms.

4 Relating dinaturality and typabililty

The result of this section is that, if $M \in DIN_\sigma$ is closed and $\beta\eta$-normal, then $\vdash M : \sigma$ is derivable in $\lambda_\to$. We will rely on the characterization of typability given by Theorem 5 and on two lemmas relating the dinaturality condition with the fitness conditions (Definition 6). In particular, it will be shown, first, that the dinaturality equation implies all fitness conditions relative to the empty path and then that, given fitness at path π, by reducing the dinaturality equation for M_π, a dinaturality equation for the subterms $M_{\pi * p}$ is obtained (whence fitness at paths $\pi * p$). Hence, simple typability for a closed λ-term is checked recursively by reducing the associated dinaturality equation.

The first lemma relates dinaturality with $1,2$-fitness:

▶ **Lemma 11.** *Let $\sigma = \sigma_1 \to \dots \to \sigma_n \to Z_u$ be a simple type and M a closed $\beta\eta$-normal λ-term satisfying:*

$$f_u\big(M(K_{\sigma_1} g_1)\dots(K_{\sigma_n} g_n)\big) \simeq_{\beta\eta} M(H_{\sigma_1} g_1)\dots(H_{\sigma_n} g_n) \tag{14}$$

for some $n \geq 1$ and λ-variables $g_1, \dots, g_n$. Then

$$M = \lambda x_1 \dots . \lambda x_h . (x_i) Q_1 \dots Q_p \tag{15}$$

for certain terms $Q_1, \dots, Q_p$, where $h \leq n$, $1 \leq i \leq h$, and $ar(\sigma_i) = p + (n - h)$.

Proof. Let $k_i = ar(\sigma_i)$ and, for $1 \leq j \leq n$, $P_j = K_{\sigma_j} g_j$ and $P'_j = H_{\sigma_j} g_j$. For a λ-term U, we let U^* (resp. U^{**}) indicate $U[P_1/x_1, \dots, P_n/x_n]$ (resp. $U[P'_1/x_1, \dots, P'_n/x_n]$). We will show that $h \leq n$. We consider two cases:

- if $h > n$ and $i > n$, then we can write $M = \lambda x_1 \dots . \lambda x_n . \lambda y_1 \dots . \lambda y_m . y_j M_1 \dots M_p$ where $m \geq 1$, and $1 \leq j \leq m$; then

$$f_u\big(MP_1 \dots P_n\big) \;\to^*_\beta\; f_u\big(\lambda y_1 \dots . \lambda y_m . y_j M_1^* \dots M_p^*\big) \tag{16}$$

  cannot be $\beta\eta$-equivalent to

$$MP'_1 \dots P'_n \;\to^*_\beta\; \lambda y_1 \dots . \lambda y_m . y_j M_1^{**} \dots M_p^{**} \tag{17}$$

- if $n \leq h$ and $i \leq n$, then we can write $M = \lambda x_1 \dots . \lambda x_n . \lambda y_1 \dots . \lambda y_m . x_i M_1 \dots M_p$ where $m \geq 0$, and $1 \leq i \leq n$; then we claim that

$$f_u\big(MP_1 \dots P_n\big) \;\to^*_\beta\; f_u\big(\lambda y_1 \dots . \lambda y_m . K_{\sigma_i} g_i M_1^* \dots M_p^*\big) \tag{18}$$

  can be $\beta\eta$-equivalent to

$$MP'_1 \dots P'_n \;\to^*_\beta\; \lambda y_1 \dots . \lambda y_m . H_{\sigma_i} g_i M_1^{**} \dots M_p^{**} \tag{19}$$

  only if $p = k_i$ and $m = 0$ (that is, $h = n$). We divide the argument in three parts:

  1. suppose $p < k_i$; then

$$MP'_1 \dots P'_n \;\to^*_\beta\; \lambda y_1 \dots . \lambda y_m . \lambda z_{k_i - p} \dots . \lambda z_{k_i} . f_u\big(g_i Q_1 \dots Q_p \dots Q_{k_i}\big) \tag{20}$$

  for some terms $Q_1, \dots, Q_{k_i}$, so it cannot be $\beta\eta$-equivalent to $f_u(MP_1 \dots P_n)$.

  2. suppose now $p > k_i$; then

$$MP'_1 \dots P'_n \;\to^*_\beta\; \lambda y_1 \dots . \lambda y_m . f_u\big(g_i Q_1 \dots Q_{k_i}\big) M_{k_i+1}^{**} \dots M_p^{**} \tag{21}$$

  so it is $\beta\eta$-equivalent to $f_u(MP_1 \dots P_n)$ only if $m = p - k_i$ and, for $1 \leq j \leq m$, $M^{**}_{k_i+j} \simeq_{\beta\eta} y_j$ and y_j is not free in $Q_1, \dots, Q_p$. We claim that $M^{**}_{k_i+j} \simeq_{\beta\eta} y_j$ only if $M_{k_i+j} \simeq_{\beta\eta} y_j$. It will follow that, if $m = p - k_i > 0$, then M is not in $\beta\eta$-normal form, against the hypothesis. To show this we will use an induction on the number of applications in M_{k_i+j}. If $M^{**}_{k_i+j} \simeq_{\beta\eta} y_j$, then M_{k_i+j} is either of the form $\lambda u_1 \dots . \lambda u_q . x_v N_1 \dots N_r$, for some $q, r \geq 0$ and $1 \leq v \leq n$, either of the form $\lambda u_1 \dots . \lambda u_q . y_v N_1 \dots N_r$, for some $q, r \geq 0$ and $1 \leq v \leq m$ or of the form $\lambda u_1 \dots . \lambda u_q . u_l N_1 \dots N_r$, for some $q, r \geq 0$ and $1 \leq l \leq q$; in the first case $M^{**}_{k_i+j}$ reduces to a term of the form $\lambda u_1 \dots . \lambda u_q . \lambda h_1 \dots . \lambda h_{q'} . f_u N'_1 \dots N'_{r'}$, for some $q', r' \geq 0$, so it cannot be $\beta\eta$-equivalent to y_j; in the second case $M^{**}_{k_i+j} \simeq_{\beta\eta} \lambda u_1 \dots . \lambda u_q . y_v N_1^{**} \dots N_r^{**}$, so it reduces to y_v only if either $q = r = 0$ and $v = j$, or $q = r$, $v = j$ and $N_k^{**} \simeq_{\beta\eta} u_k$, for $1 \leq k \leq q$; in this latter case we can apply the induction hypothesis on the terms N_k: $N_k^{**} \simeq_{\beta\eta} u_k$ only if $N_k \simeq_{\beta\eta} u_k$. In both cases we conclude that $M^{**}_{k_i+j} \simeq_{\beta\eta} y_j$ only if $M_{k_i+j} \simeq_{\beta\eta} y_j$; in the third case $M^{**}_{k_i+j} \simeq_{\beta\eta} \lambda u_1 \dots . \lambda u_q . u_l N_1^{**} \dots N_r^{**}$ cannot be $\beta\eta$-equivalent to y_l.

  3. Finally, suppose $m \geq 1$; then $MP'_1 \dots P'_n$ reduces to $\lambda y_1 \dots . \lambda y_m . f_u\big(g_i Q_1 \dots Q_p\big)$, so it cannot be $\beta\eta$-equivalent to $f_u(MP_1 \dots P_n)$.

We have so far shown that $h \leq n$ and moreover that, if $h = n$, then $p = k_i$. Let now $M = \lambda x_1 \dots . \lambda x_h . x_i M_1 \dots M_p$, with $h < n$ and $i \leq h$. We prove that $k_i = p + (n - h)$: if $p + (n - h) < k_i$, then $MP'_1 \dots P'_n$ reduces to $\lambda z_{k_i - d} \dots . \lambda z_{k_i} . f_u\big(g_i Q_1 \dots Q_p \dots Q_{k_i}\big)$, where $d = p + (n - h)$, for some terms $Q_1, \dots, Q_{k_i}$, so it cannot be $\beta\eta$-equivalent to $f_u(MP_1 \dots P_n)$. If $p + (n - h) > k_i$, then $MP'_1 \dots P'_n$ reduces to $f_u(g_i Q_1 \dots Q_{k_i}) M_{p-e}^{**} \dots M_p^{**}$, where $e = k_i - (n - h)$, so it cannot be $\beta\eta$-equivalent to $f_u(MP_1 \dots P_n)$. ◀

Lemma 11 says that a closed $\beta\eta$-normal dinatural λ-term is $1,2$-fit at ϵ. Indeed, from $h \leq ar(\sigma)$ it follows that $\mathtt{var}(\epsilon, i)$ is defined for all $(\epsilon, i) \in Var(M)$ (and $\mathtt{subt}(\epsilon)$ is always defined); moreover, we have $ar(\sigma_{\mathtt{var}(h(\epsilon))}) = ar(\sigma_i) = p+(n-h) = p_M(\epsilon)+ar(\sigma_{\mathtt{subt}(\epsilon)})-ar(N_\epsilon)$. The next lemma relates dinaturality and 3-fitness:

▶ **Lemma 12.** *For all simple types σ, τ the equation below*

$$H_\sigma(K_\tau g) \simeq_{\beta\eta} K_\sigma(H_\tau g) \tag{22}$$

holds if and only if $\sigma = \tau$.

Proof. We argue by induction on σ: if $\sigma = Z_u$ and $\tau = \tau_1 \to \cdots \to \tau_k \to Z_v$, for some $k \geq 0$, then $H_\sigma(K_\tau g) \simeq_{\beta\eta} K_\sigma(H_\tau g)$ becomes

$$\begin{aligned} H_{Z_u}(K_\tau g) \simeq_\beta f_u(K_\tau g) \simeq_\beta f_u\big(\lambda h_1.\ldots.\lambda h_k.g(H_{\tau_1}h_1)\ldots(H_{\tau_k}h_k)\big) \simeq_{\beta\eta} \\ \lambda h_1.\ldots.\lambda h_k.f_v\big(g(K_{\tau_1}k_1)\ldots(K_{\tau_k}h_k)\big) \simeq_\beta H_\tau g \simeq_\beta K_{Z_v}(H_\tau g) \end{aligned} \tag{23}$$

and the central equation holds if and only if $k = 0$ and $u = v$, i.e. if and only if $\sigma = \tau = Z_u$.

For the induction step, let $\sigma = \sigma_1 \to \cdots \to \sigma_k \to Z_u$ and $\tau = \tau_1 \to \cdots \to \tau_{k'} \to Z_v$. We can suppose w.l.o.g. $k' = k + d$; now $H_\sigma(K_\tau g)$ reduces to

$$\lambda h_1.\ldots.\lambda h_k.f_u\Big(\lambda h_{k+1}.\ldots.\lambda h_{k'}.g(K_{\sigma_1}(H_{\tau_1}h_1))\ldots(K_{\sigma_k}(H_{\tau_k}h_k))(H_{\tau_{k+1}}h_{k+1})\ldots(H_{\tau_{k'}}h_{k'})\Big) \tag{24}$$

and $K_\sigma(H_\tau g)$ reduces to

$$\lambda h_1.\ldots.\lambda h_k.\lambda h_{k+1}.\ldots.\lambda h_{k'}.f_v\Big(g(H_{\sigma_1}(K_{\tau_1}h_1))\ldots(H_{\sigma_k}(K_{\tau_k}h_k))(K_{\tau_{k+1}}h_{k+1})\ldots(K_{\tau_{k'}}h_{k'})\Big) \tag{25}$$

and the two terms are $\beta\eta$-equivalent if and only if $d = 0$, $u = v$ and, for all $1 \leq i \leq k$, $\sigma_i = \tau_i$ (by induction hypothesis), i.e if and only if $\sigma = \tau$. ◀

Lemma 12 says that a closed $\beta\eta$-normal dinatural λ-term is 3-fit at ϵ: if $M \in DIN_\sigma$ and $M = \lambda x_1.\ldots.\lambda x_h.x_i M_1 \ldots M_p$, then, by Lemma 11, $h \leq n$ and $k_i = p + (n - h)$; so, equation 9 reduces to

$$\begin{aligned} f_u\Big(g_i(H_{\sigma_{i1}}M_1^*)\ldots(H_{\sigma_{ip}}M_p^*)(H_{\sigma_{i(p+1)}}(K_{\sigma_{h+1}}g_{h+1}))\ldots(H_{\sigma_{i(ar(\sigma_i))}}(K_{\sigma_n}g_n))\Big) \\ \simeq_{\beta\eta} f_u\Big(g_i(K_{\sigma_{i1}}M_1^{**})\ldots(K_{\sigma_{ip}}M_p^{**})(K_{\sigma_{i(p+1)}}(H_{\sigma_{h+1}}g_{h+1}))\ldots(K_{\sigma_{i(ar(\sigma_i))}}(H_{\sigma_n}g_n))\Big) \end{aligned} \tag{26}$$

where M_j^*, M_j^{**} are as in the proof of Lemma 11. Now, by Lemma 12, we get that, for $1 \leq j \leq n - h$, $\sigma_{h+j} = \sigma_{i(p+j)}$, i.e. $\sigma_{\mathtt{subt}(\epsilon)*(ar(M)+j)} = \sigma_{\mathtt{var}(h(\epsilon))*(p_M(\epsilon)+j)}$.

We will now show that, for $M \in DIN_\sigma$ closed and $\beta\eta$-normal, p-fitness can be extended to all paths $\pi \in \mathcal{T}(M)$: by computing the dinaturality equation, and exploiting lemmas 11 and 12, we obtain a dinaturality equation for all subterms M_π.

▶ **Proposition 13.** *Let σ be a simple type and M be a closed $\beta\eta$-normal λ-term in DIN_σ. Then, for every $\pi \in \mathcal{T}(M)$, M_π is p-fit for $\sigma_{\mathtt{subt}(\pi)}$, for $p = 1, 2, 3$.*

Proof. We argue by induction on the number of applications in M. If $M = \lambda x_1.\ldots.\lambda x_h.x_i$, then $\mathcal{T}(M) = \{\epsilon\}$ so the claim holds since $M_\epsilon = M$ is p-fit for $p = 1, 2, 3$ by lemmas 11 and 12. Let then $M = \lambda x_1.\ldots.\lambda x_h.x_i M_1 \ldots M_p$ for some $p \geq 1$ and let $N_j = \lambda x_1.\ldots.\lambda x_n.M_j$ and

$\tau_j = \sigma_1 \to \cdots \to \sigma_n \to \sigma_{ij}$. Again, by Lemma 11 and 12, M_ϵ is p-fit to σ_ϵ, for $p = 1, 2, 3$. Let then $\pi = j * \pi' \in \mathcal{T}(M)$, with $\pi' \in \mathcal{T}(N_j)$ and $1 \leq j \leq p$. Since, by Lemma 11, $h \leq n$ and $k_i = p + (n - h)$ (where k_i is $ar(\sigma_i)$), equation 9 reduces to

$$f_u\big(K_{\sigma_i} g_i M_1^* \dots M_p^* (K_{\sigma_{p+1}} g_{h+1}) \dots (K_{\sigma_n} g_n)\big) \simeq_{\beta\eta} H_{\sigma_i} g_i M_1^{**} \dots M_p^{**} (H_{\sigma_{p+1}} g_{h+1}) \dots (H_{\sigma_n} g_n) \tag{27}$$

where M_j^*, M_j^{**} are as in the proof of Lemma 11; this in turn reduces to

$$\begin{aligned} & f_u\Big(g_i\big(H_{\sigma_{i1}} M_1^*\big) \dots \big(H_{\sigma_{ip}} M_p^*\big)\big(H_{\sigma_{i(p+1)}}(K_{\sigma_{p+1}} g_{p+1})\big) \dots \big(H_{\sigma_{i(ar(\sigma_i))}}(K_{\sigma_n} g_n)\big)\Big) \\ \simeq_{\beta\eta}\ & f_u\Big(g_i\big(K_{\sigma_{i1}} M_1^{**}\big) \dots \big(K_{\sigma_{ip}} M_p^{**}\big)\big(K_{\sigma_{i(p+1)}}(H_{\sigma_{p+1}} g_{p+1})\big) \dots \big(K_{\sigma_{i(ar(\sigma_i))}}(H_{\sigma_n} g_n)\big)\Big) \end{aligned} \tag{28}$$

By Lemma 12, $\sigma_{i(p+j)} = \sigma_{p+j}$, for all $1 \leq j \leq n - h$. From equation 28 we obtain then $H_{\sigma_{ij}} M_j^* \simeq_{\beta\eta} K_{\sigma_{ij}} M_j^{**}$. Now we have that

$$\begin{aligned} & f_v\Big(N_j (K_{\sigma_1} g_1) \dots (K_{\sigma_h} g_h)(K_{\sigma_{ij1}} h_1) \dots (K_{\sigma_{ijd}} h_d)\Big) \simeq_\beta H_{\sigma_{ij}} M_j^* \\ \simeq_{\beta\eta}\ & K_{\sigma_{ij}} M_j^{**} \simeq_\beta N_j (H_{\sigma_1} g_1) \dots (H_{\sigma_h} g_h)(H_{\sigma_{ij1}} h_1) \dots (H_{\sigma_{ijd}} h_d) \end{aligned} \tag{29}$$

where $d = ar(\sigma_{ij})$, which implies that $N_j \in DIN_{\tau_j}$. We can then apply the induction hypothesis to N_j: by letting $h_j : \mathcal{T}(N_j) \to Var(N_j), \mathtt{var}_j : Var(N_j) \to \mathcal{T}(\tau_j), \mathtt{subt}_j : \mathcal{T}(N_j) \to \mathcal{T}(\tau_j)$ indicate the $h, \mathtt{var}, \mathtt{subt}$ functions, respectively, defined for N_j, we have that, for any $\pi' \in \mathcal{T}(N_j)$, $(N_j)_{\pi'}$ is p-fit to $(\tau_j)_{\mathtt{subt}_j(\pi')}$, for $p = 1, 2, 3$.

In order to prove that $M_\pi = (N_j)_{\pi'}$ is p-fit to $\sigma_{\mathtt{subt}(\pi)}$, it is enough to verify that, for all $\lambda \in \mathcal{T}(N_j)$, the following equalities hold:

$$(\tau_j)_{\mathtt{var}_j(h_j(\lambda))} = \sigma_{\mathtt{var}(h(j*\lambda))} \qquad (\tau_j)_{\mathtt{subt}_j(\lambda)} = \sigma_{\mathtt{subt}(j*\lambda)} \tag{30}$$

We postpone this technical verification to Appendix A. ◀

▶ **Theorem 14.** *Let σ be a simple type and $M \in DIN_\sigma$ be closed and $\beta\eta$-normal. Then $\vdash M : \sigma$ is derivable.*

Proof. By applying Proposition 13 and Theorem 5. ◀

5 Parametricity in the $\beta\eta$-stable semantics

In this section we recall the interpretation of $\lambda_\to$ by means of $\beta\eta$-stable sets of λ-terms (see [7]) and we define a variant of Reynolds' binary parametricity in this frame.

Let $S_{\beta\eta}$ be the set of all sets $s \subseteq \Lambda$ stable by $\beta\eta$-equivalence. By an interpretation we mean any function $\mathcal{M} : \mathcal{V} \to S_{\beta\eta}$. For any $T, U \in S_{\beta\eta}$, we let $T \to U := \{M \in \Lambda | N \in T \Rightarrow MN \in U\} \in S_{\beta\eta}$.

▶ **Definition 15.** For any simple type σ and assignment $\mathcal{M} : \mathcal{V} \to \mathcal{S}_{\beta\eta}$, we define an interpretation $|\sigma|_\mathcal{M} \in S_{\beta\eta}$:

$$\begin{aligned} |Z_u|_\mathcal{M} &:= \mathcal{M}(Z_u) \\ |\sigma \to \tau|_\mathcal{M} &:= |\sigma|_\mathcal{M} \to |\tau|_\mathcal{M} \end{aligned} \tag{31}$$

For any simple type σ, we define its *uniform interpretation* $|\sigma| := \bigcap_\mathcal{M} |\sigma|_\mathcal{M}$. A closed λ-term M will be said *interpretable (for σ)* when $M \in |\sigma|$.

Interpretable terms are normalizable:

▶ **Proposition 16.** *If a closed λ-term $M \in |\sigma|$, then there exists $M' \simeq_{\beta\eta} M$ such that M' is in $\beta\eta$-normal form.*

Proof. Let us call a λ-term idle if it is $\beta\eta$-equivalent to a $\beta\eta$-normal term which does not begin with a λ. Let $\mathcal{M}$ be the interpretation such that, for all Z_u, $\mathcal{M}(Z_u)$ is the set of all idle terms. Clearly, for all σ and variable x, $x \in |\sigma|_{\mathcal{M}}$. From $M \in |\sigma|_{\mathcal{M}}$, it follows then that $Mx_1 \dots x_n$ is idle (where $\sigma = \sigma_1 \to \dots \to \sigma_n \to Z_u$), hence there exists $M' \simeq_{\beta\eta} Mx_1 \dots x_n$ $\beta\eta$-normal. We have then $M \simeq_{\beta\eta} \lambda x_1 . \dots . \lambda x_n . Mx_1 \dots x_n \simeq_{\beta\eta} \lambda x_1 . \dots . \lambda x_n . M'$. Now $\lambda x_1 . \dots . \lambda x_n . M'$ is either $\beta\eta$-normal, and in this case we are done, or it is of the form $\lambda x_1 . \dots . \lambda x_h . \lambda x_{h+1} . \dots . \lambda x_n . M'' x_{h+1} \dots x_n$, for some $1 \leq h < n$, with $x_{h+1}, \dots, x_n$ not free in M''. In this case $M \simeq_{\beta\eta} \lambda x_1 . \dots . \lambda x_h . M''$ which is $\beta\eta$-normal. ◀

We recall the completeness theorem for the $\beta\eta$-stable semantics:

▶ **Theorem 17** ([6, 3])**.** *Let M be a closed λ-term; if $M \in |\sigma|$, then there exists $M' \simeq_{\beta\eta} M$ such that $\vdash M' : \sigma$ is derivable.*

The result in [3] is actually more general, as it holds for *positive types* in Krivine's system $\mathcal{AF}2$, i.e. second order types built from atomic types of the form $X(t_1, \dots, t_n)$, where the t_i are first-order terms, implication, first order quantifiers and second order universal quantifiers (the latter occurring only positively).

In the following we will not use Farkh and Nour's theorem, as we want to prove parametricity and dinaturality by semantic means, that is, without relying on typability.

We introduce the notion of $\beta\eta$-stable relations:

▶ **Definition 18.** Let $s, t \subseteq S_{\beta\eta}$. A *$\beta\eta$-stable relation* is a binary relation $r \subseteq s \times t$ such that for all $M, M' \in s, N, N' \in t$, if $(M, N) \in r$ (what we will note by $M \; r \; N$), $M \simeq_{\beta\eta} M'$ and $N \simeq_{\beta\eta} N'$, then $M' \; r \; N'$.

▶ **Definition 19.** If $r \subseteq s \times t$ and $r' \subseteq s' \times t'$ are $\beta\eta$-stable relations, the $\beta\eta$-stable relation $r \to r' \subseteq (s \to s') \times (t \to t')$ is defined by $P \; (r \to r') \; Q$ if for all $M \in s, N \in t$, if $M \; r \; N$ then $(PM) \; r' \; (QN)$.

In the following lines we reformulate Reynolds' parametricity with respect to $\beta\eta$-stable relations.

▶ **Definition 20** (relation assignment)**.** Let $\mathcal{M}_1, \mathcal{M}_2 : \mathcal{V} \to S_{\beta\eta}$ be two interpretations. A *relation assignment R over $\mathcal{M}_1$ and $\mathcal{M}_2$* is a map associating, with any simple type σ, a $\beta\eta$-stable relation $R[\sigma] \subseteq |\sigma|_{\mathcal{M}_1} \times |\sigma|_{\mathcal{M}_2}$ such that, if $\sigma = \sigma_1 \to \sigma_2$, then $R[\sigma] = R[\sigma_1] \to R[\sigma_2]$.

▶ **Definition 21** (parametricity)**.** Let σ be a simple type and M be a closed λ-term. M is *parametric in σ* if, for all interpretations $\mathcal{M}_1, \mathcal{M}_2$ and relation assignment R over $\mathcal{M}_1, \mathcal{M}_2$, $M \; R[\sigma] \; M$.

▶ **Example 22.** Let σ be $(Z_u \to Z_u) \to (Z_u \to Z_u)$ as in Example 9. Parametricity in σ for a closed λ-term M corresponds to the fact that, for any two $\mathcal{M}_1, \mathcal{M}_2 : \mathcal{V} \to S_{\beta\eta}$ and relation assignment R over $\mathcal{M}_1$ and $\mathcal{M}_2$, and for any $F_0 \in |Z_u|_{\mathcal{M}_1}, G_0 \in |Z_u|_{\mathcal{M}_2}$ and $F_1 \in |Z_u \to Z_u|_{\mathcal{M}_1}, G_1 \in |Z_u \to Z_u|_{\mathcal{M}_2}$, if $F_0 \; R[Z_u] \; G_0$ and $F_1 \; R[Z_u \to Z_u] \; G_1$, then $MF_1F_0 \; R[Z_u] \; MG_1G_0$.

If M is a closed term of type σ, the $\beta\eta$-normal form of M is of the form $\lambda y . \lambda x . y^n x$, for some $n \geq 0$. Then we can verify that M is parametric in σ. Let F_0, G_0, F_1, G_1 be as

before; we have that $MF_1F_0 \simeq_{\beta\eta} F_1^nF_0$ and $MG_1G_0 \simeq_{\beta\eta} G_1^nG_0$. Now we can argue by induction on n: if $n = 0$, then $F_1^nF_0 = F_0\ R[Z_u]\ G_0 = G_1^nG_0$ by hypothesis; if $n = k+1$, then, by induction hypothesis, $F_1^kF_0\ R[Z_u]\ G_1^kG_0$ and, by hypothesis, $F_1\ R[Z_u \to Z_u]\ G_1$, that is, for all U, V such that $U\ R[Z_u]\ V$, $F_1U\ R[Z_u]\ G_1V$, so we can conclude $F_1^nF_0 = F_1(F_1^kF_0)\ R[Z_u]\ G_1(G_1^kG_0) = G_1^nG_0$.

The theorem below, known as Reynolds' *abstraction theorem* ([10]), generalizes this example to an arbitrary closed simply-typed λ-term.

▶ **Theorem 23.** *Let σ be a simple type and M be a closed λ-term. If $\vdash M : \sigma$, then M is parametric in σ.*

Proof. The proof is exactly as in [10]. ◀

In the next section we will prove that closed interpretable λ-terms are parametric, that is, we will prove parametricity for a term M without relying on a typing of M but rather on the basis of a semantic property of M.

6 The parametricity theorem

In this section we prove that interpretable closed λ-terms are parametric (in the sense of Definition 21) by a semantic argument. The proof exploits the idea, appearing in the completeness proofs for the $\beta\eta$-stable semantics in [6, 3], of defining an infinite context Γ^∞ made of declarations $(x_i : \tau_i)$, given an enumeration $(x_i)_{i\in\mathbb{N}}$ of λ-variables and an enumeration $(\tau_i)_{i\in\mathbb{N}}$ of simple types, such that each type receives infinitely many indices. However, the infinite context Γ^∞ will not be used, as in those proofs, to define a contextual typability relation, but rather to define contextual notions of interpretation and relation assignment. In particular, a special interpretation $\mathcal{M}_P$ will be defined such that, for any simple type σ and λ-term P, if $P \in |\sigma|_{\mathcal{M}_P}$, then for any two interpretations and relation assignment over them, P is contextually related to P relative to σ.

▶ **Theorem 24** (parametricity). *For any simple type σ and closed term M, if $M \in |\sigma|$, then M is parametric in σ.*

Proof. Let $(x_i)_{i\in\mathbb{N}}$ be an enumeration of the λ-variables and $(\tau_i)_{i\in\mathbb{N}}$ an enumeration of simple types such that every type has infinitely many indices. Let $\Gamma^\infty = \{x_i : \tau_i \mid i \in \mathbb{N}\}$ and, for every term M, Γ_M be the context made by restricting Γ^∞ to the free variables occurring in M.

We first define contextual interpretations and relations:

- given $\mathcal{M} : \mathcal{V} \to S_{\beta\eta}$, a λ-term P and a simple type τ, the statement $P \in |\Gamma^\infty \vdash \tau|_{\mathcal{M}}$ holds when, by letting $x_{i_1}, \dots, x_{i_n}$ be the free variables of P, for every $Q_1 \in |\tau_{i_1}|_{\mathcal{M}}, \dots, Q_n \in |\tau_{i_n}|_{\mathcal{M}}$, $P[Q_1/x_{i_1}, \dots, Q_n/x_{i_n}] \in |\tau|_{\mathcal{M}}$;
- given $\mathcal{M}_1, \mathcal{M}_2 : \mathcal{V} \to S_{\beta\eta}$ and R a relation assignment over $\mathcal{M}_1, \mathcal{M}_2$, λ-terms P, Q and a simple type τ, the statement $P\ R[\Gamma^\infty \vdash \tau]\ Q$ holds when $P \in |\Gamma^\infty \vdash \tau|_{\mathcal{M}_1}$, $Q \in |\Gamma^\infty \vdash \tau|_{\mathcal{M}_2}$ and, by letting $x_{i_1}, \dots, x_{i_n}$ be the free variables of P and Q, for every $F_j \in |\tau_{i_j}|_{\mathcal{M}_1}, G_j \in |\tau_{i_j}|_{\mathcal{M}_2}$ such that $F_j\ R[\tau_{i_j}]\ G_j$ for $1 \le j \le n$, we have

$$P[F_1/x_{i_1}, \dots, F_n/x_{i_n}]\ R[\tau]\ Q[G_1/x_{i_1}, \dots, G_n/x_{i_n}] \tag{32}$$

Let now $\mathcal{M}_P : \mathcal{V} \to S_{\beta\eta}$ be the assignment such that, for all $Z_u \in \mathcal{V}$, $\mathcal{M}_P(Z_u)$ is the $\beta\eta$-closure of the set of λ-terms P such that $P \in |\Gamma^\infty \vdash Z_u|$ and for all $\mathcal{M}_1, \mathcal{M}_2 : \mathcal{V} \to S_{\beta\eta}$ and relation assignment R over $\mathcal{M}_1, \mathcal{M}_2$, $P\ R[\Gamma^\infty \vdash Z_u]\ P$.

We claim that, for any simple type σ, the following hold:

1. if $P \in |\sigma|_{\mathcal{M}_P}$, then, for any $\mathcal{M}_1, \mathcal{M}_2 : \mathcal{V} \to S_{\beta\eta}$ and relation assignment R over them, $P\ R[\Gamma^\infty \vdash \sigma]\ P$;
2. for every variable x_i such that $\tau_i = \sigma$, $x_i \in |\sigma|_{\mathcal{M}_P}$.

We argue for both 1. and 2. by induction on σ. If $\sigma = Z_u$, then, by definition, any $P \in |\sigma|_{\mathcal{M}_P}$ is such that, for any $\mathcal{M}_1, \mathcal{M}_2 : \mathcal{V} \to S_{\beta\eta}$ and relation assignment R over them, $P\ R[\Gamma^\infty \vdash \sigma]\ P$ holds, so claim 1. holds; moreover, if $\tau_i = Z_u$, then, for any $\mathcal{M}_1, \mathcal{M}_2 : \mathcal{V} \to S_{\beta\eta}$ and relation assignment R over $\mathcal{M}_1, \mathcal{M}_2$, $x_i\ R[\Gamma^\infty \vdash Z_u]\ x_i$: if $F \in \mathcal{M}_1(Z_u), G \in \mathcal{M}_2(Z_u)$ and $F\ R[Z_u]\ G$, then $x_i[F/x_i] = F_i\ R[Z_u]\ G_i = x_i[G/x_i]$. Hence claim 2. holds too.

Let now $\sigma = \sigma_1 \to \sigma_2$. By induction hypothesis, for all $i \geq 0$, if $\tau_i = \sigma_1$ then $x_i \in |\sigma_1|_{\mathcal{M}_P}$; let then $P \in |\sigma|_{\mathcal{M}_P}$ and choose an index i, with $\tau_i = \sigma_1$, such that x_i does not occur free in P; we have that $Px_i \in |\sigma_2|_{\mathcal{M}_P}$ hence, by induction hypothesis, for any $\mathcal{M}_1, \mathcal{M}_2 : \mathcal{V} \to S_{\beta\eta}$ and relation assignment R over them, $(Px_i)\ R[\Gamma^\infty \vdash \sigma_2]\ (Px_i)$; let then $\mathcal{M}_1, \mathcal{M}_2 : \mathcal{V} \to S_{\beta\eta}$ and R be a relation assignment over them; by letting $\Gamma_P = \{x_{i_1} : \tau_{i_1}, \dots, x_{i_n} : \tau_{i_n}\}$, suppose $F_j \in |\tau_{i_j}|_{\mathcal{M}_1}, G_j \in |\tau_{i_j}|_{\mathcal{M}_2}$ are such that $F_j\ R[\tau_{i_j}]\ G_j$, for $1 \leq j \leq n$, and moreover suppose $F \in |\sigma_1|_{\mathcal{M}_1}, G \in |\sigma_1|_{\mathcal{M}_2}$ are such that $F\ R[\sigma_1]\ G$; then, since $\Gamma_{Px_i} = \{x_{i_1} : \tau_{i_1}, \dots, x_{i_n} : \tau_{i_n}, x_i : \sigma_1\}$ (remark that $x_i \neq x_{i_1}, \dots, x_{i_n}$ as x_i has been chosen not to occur free in P), we have

$$\begin{aligned}
&\big((\lambda x_i.Px_i)[F_1/x_{i_1}, \dots, F_n/x_{i_n}]\big)F \simeq_{\beta\eta} (Py)[F_1/x_{i_1}, \dots, F_n/x_{i_n}][F/y] \\
&\simeq_{\beta\eta} (Px_i)[F_1/x_{i_1}, \dots, F_n/x_{i_n}, F/x_i]\ R[\sigma_2]\ (Px_i)[G_1/x_{i_1}, \dots, G_n/x_{i_n}, G/x_i] \\
&\simeq_{\beta\eta} (Py)[G_1/x_{i_1}, \dots, G_n/x_{i_n}][G/y] \simeq_{\beta\eta} \big((\lambda x_i.Px_i)[G_1/x_{i_1}, \dots, G_n/x_{i_n}]\big)G
\end{aligned} \tag{33}$$

where we can suppose y not occurring free in any of the F_j and G_j. We conclude then that $P \simeq_{\beta\eta} \lambda x_i.Px_i\ R[\Gamma^\infty \vdash \sigma]\ \lambda x_i.Px_i \simeq_{\beta\eta} P$, so we proved claim 1.

To prove claim 2., suppose x_i is a variable such that $\tau_i = \sigma$. Let $\sigma = \sigma_1 \to \dots \to \sigma_n \to Z_v$ for some $n \geq 1$ and $Q_1, \dots, Q_n$ be terms such that $Q_j \in |\sigma_j|_{\mathcal{M}_P}$, for $1 \leq j \leq n$; by induction hypothesis, for all $\mathcal{M}_1, \mathcal{M}_2 : \mathcal{V} \to S_{\beta\eta}$ and relation assignment R over $\mathcal{M}_1, \mathcal{M}_2$, $Q_j\ R[\Gamma^\infty \vdash \sigma_j]\ Q_j$, for $1 \leq j \leq n$. Let $\mathcal{M}_1, \mathcal{M}_2 : \mathcal{V} \to S_{\beta\eta}$, and R be a relation assignment over $\mathcal{M}_1, \mathcal{M}_2$; moreover let $\{x_i : \tau_i\} \cup \bigcup_i \Gamma_{Q_i}$ be the set $\{x_{i_1} : \tau_{i_1}, \dots, x_{i_r} : \tau_{i_r}\}$ where $i = i_p$ for some fixed $1 \leq p \leq r$; given terms $F_1, G_1, \dots, F_r, G_r$ such that $F_l \in |\tau_{i_l}|_{\mathcal{M}_1}, G_l \in |\tau_{i_l}|_{\mathcal{M}_2}$ and $F_l\ R[\tau_{i_l}]\ G_l$ all hold for $1 \leq l \leq r$, we have that

$$(x_iQ_1 \dots Q_n)\theta_1 \simeq_{\beta\eta}\ F_{i_p}(Q_1\theta_1) \dots (Q_n\theta_1) \tag{34}$$

and

$$(x_iQ_1 \dots Q_n)\theta_2 \simeq_{\beta\eta}\ G_{i_p}(Q_1\theta_2) \dots (Q_n\theta_2) \tag{35}$$

where θ_1 (resp. θ_2) is the substitution $[F_1/x_{i_1}, \dots, F_r/x_{i_r}]$ (resp. $[G_1/x_{i_1}, \dots, G_r/x_{i_r}]$).

Now, from the induction hypothesis (which implies that $Q_j\theta_1\ R[\sigma_j]\ Q_j\theta_2$) and the fact that $F_{i_p}\ R[\sigma]\ G_{i_p}$, it follows that $x_iQ_1 \dots Q_n\ R[\Gamma^\infty \vdash Z_u]\ x_iQ_1 \dots Q_n$. We deduce that $x_iQ_1 \dots Q_n \in |Z_v|_{\mathcal{M}_P}$, that is, $x_i \in |\sigma|_{\mathcal{M}_P}$ and claim 2. is proved.

Finally, let M be closed and interpretable. Then $M \in |\sigma|_{\mathcal{M}_P}$, so, for every $\mathcal{M}_1, \mathcal{M}_2 : \mathcal{V} \to S_{\beta\eta}$ and relation assignment R over $\mathcal{M}_1, \mathcal{M}_2$, $M\ R[\Gamma^\infty \vdash \sigma]\ M$, that is $M\ R[\sigma]\ M$, as M is closed. ◀

7 From parametricity to dinaturality

In this section we adapt the well-known fact that parametricity implies dinaturality ([9]) to the frame described in the last sections, obtaining a semantic argument that interpretable closed λ-terms are dinatural.

Let $\mathcal{M}_1, \mathcal{M}_2 : \mathcal{V} \to S_{\beta\eta}$ be given, for any Z_u, by $\mathcal{M}_1(Z_u) = \Lambda$ and $\mathcal{M}_2(Z_u) = f_u\Lambda$, where $f_u\Lambda$ is the $\beta\eta$-closure of the set of all λ-terms of the form f_uQ, for some $Q \in \Lambda$. Let moreover the relation assignment R^f over $\mathcal{M}_1, \mathcal{M}_2$ be defined, for any Z_u, by $P\ R[Z_u]\ Q$, if $f_uP \simeq_{\beta\eta} Q$. Let $\mathcal{V}_1, \mathcal{V}_2$ be as in Section 3 and $\mathcal{N} : \mathcal{V}_1 \cup \mathcal{V}_2 \to S_{\beta\eta}$ be the interpretation such that, for all $u \geq 0$, $\mathcal{N}(X_u) = \mathcal{M}_1(Z_u)$ and $\mathcal{N}(Y_u) = \mathcal{M}_2(Z_u)$.

▶ **Proposition 25.** *For any type σ,*

$$\begin{array}{ll} H_\tau \in |\tau_Y^X \to \tau_Y|_{\mathcal{N}} & K_\tau \in |\tau_Y \to \tau_X^Y|_{\mathcal{N}} \\ K_\tau \in |\tau_Y^X \to \tau_X|_{\mathcal{N}} & H_\tau \in |\tau_X \to \tau_X^Y|_{\mathcal{N}} \end{array} \tag{36}$$

Proof. We argue by induction on τ: if $\tau = Z_u$, then $H_\tau = f_u \in |X_u \to Y_u|_{\mathcal{N}}$ and $K_\tau = \lambda x.x \in |X_u \to X_u|_{\mathcal{N}}, |Y_u \to Y_u|_{\mathcal{N}}$. If $\tau = \tau_1 \to \cdots \to \tau_m \to Z_u$ for some $m \geq 1$, then let $E \in |\tau_Y^X|_{\mathcal{N}}, F \in |\tau_X|_{\mathcal{N}}, G \in |\tau_Y|_{\mathcal{N}}$ and $E_i \in |(\tau_i)_Y^X|_{\mathcal{N}}, F_i \in |(\tau_i)_X|_{\mathcal{N}}, G_i \in |(\tau_i)_Y|_{\mathcal{N}}$, for $1 \leq i \leq m$. Then, by induction hypothesis we have

$$\begin{array}{ll} H_{\tau_i}E_i \in |(\tau_i)_Y|_{\mathcal{N}} & K_{\tau_i}G_i \in |(\tau_i)_X^Y|_{\mathcal{N}} \\ K_{\tau_i}E_i \in |(\tau_i)_X|_{\mathcal{N}} & H_{\tau_i}F_i \in |(\tau_i)_X^Y|_{\mathcal{N}} \end{array} \tag{37}$$

from which we obtain

$$H_\tau EG_1 \dots G_n \simeq_\beta \quad f_u\big(E(K_{\tau_1}G_1)\dots(K_{\tau_n}G_n)\big) \in |Y|_{\mathcal{N}} \tag{38}$$

$$H_\tau FE_1 \dots E_n \simeq_\beta \quad f_u\big(F(K_{\tau_1}E_1)\dots(K_{\tau_n}E_n)\big) \in |Y|_{\mathcal{N}} \tag{39}$$

$$K_\tau EF_1 \dots F_n \simeq_\beta \quad E(H_{\tau_1}F_1)\dots(H_{\tau_n}F_n) \in |X|_{\mathcal{N}} \tag{40}$$

$$K_\tau GE_1 \dots E_n \simeq_\beta \quad G(H_{\tau_1}E_1)\dots(H_{\tau_n}E_n) \in |Y|_{\mathcal{N}} \tag{41}$$

exploiting the fact that $\tau_Y^X = (\tau_1)_X^Y \to \cdots \to (\tau_n)_X^Y \to X_u$. ◀

For any λ-variable $g \neq z$, $H_\tau g \in |\tau_Y|_{\mathcal{N}} = |\tau|_{\mathcal{M}_2}$ and $K_\tau g \in |\tau_X|_{\mathcal{N}} = |\tau|_{\mathcal{M}_1}$. Hence we can ask whether $H_\tau g$ and $K_\tau g$ are related under the relation assignment R^f. This is shown by the proposition below:

▶ **Proposition 26.** *For all simple type σ and variable g,*
(i) *$(K_\sigma g)\ R^f[\sigma]\ (H_\sigma g)$;*
(ii) *if $P \in |\sigma|_{\mathcal{M}_1}, Q \in |\sigma|_{\mathcal{M}_2}$ and $P\ R^f[\sigma]\ Q$, then $H_\sigma P \simeq_{\beta\eta} K_\sigma Q$.*

Proof. We prove both fact simultaneously by induction on σ. If $\sigma = Z_u$ then $K_\sigma g \simeq_\beta g\ R^f\ f_u g \simeq_\beta H_\sigma g$; moreover, if $P\ R^f[Z_u]\ Q$, then $H_\sigma P \simeq_\beta f_uP \simeq_{\beta\eta} Q \simeq_\beta K_\sigma Q$.

If $\sigma = \sigma_1 \to \cdots \to \sigma_n \to Z_u$ for some $n \geq 1$, then suppose $P_i \in |\sigma_i|_{\mathcal{M}_1}, Q_i \in |\sigma_i|_{\mathcal{M}_2}$ and $P_i\ R[\sigma_i]\ Q_i$, for all $1 \leq i \leq n$; then

$$P := K_\sigma gP_1 \dots P_n \simeq_\beta g(H_{\sigma_1}P_1)\dots(H_{\sigma_n}P_n) \tag{42}$$

is related to

$$Q := H_\sigma gQ_1 \dots Q_n \simeq_\beta f_u\Big(g(K_{\sigma_1}Q_1)\dots(K_{\sigma_n}Q_n)\Big) \tag{43}$$

Indeed, by induction hypothesis, $H_{\sigma_i}P_i \simeq_{\beta\eta} K_{\sigma_i}Q_i$, hence $g(K_{\sigma_1}P_1)\dots(K_{\sigma_n}P_n) \simeq_{\beta\eta} g(H_{\sigma_1}Q_1)\dots(H_{\sigma_n}Q_n)$ so $f_uP \simeq_{\beta\eta} Q$.

Suppose now $P \in |\sigma|_{\mathcal{M}_1}, Q \in |\sigma|_{\mathcal{M}_2}$ and $P\ R^f[\sigma]\ Q$, then let

$$P' := H_\sigma Pg_1 \dots g_n \simeq_\beta f_u\Big(P(K_{\sigma_1}g_1)\dots(K_{\sigma_n}g_n)\Big) \tag{44}$$

and

$$Q' := K_\sigma Q g_1 \dots g_n \simeq_\beta Q(H_{\sigma_1} g_1) \dots (H_{\sigma_n} g_n) \tag{45}$$

By induction hypothesis $(K_{\sigma_i} g_i)\ R^f[\sigma_i]\ (H_{\sigma_i} g_i)$ hence, by hypothesis $P(K_{\sigma_1} g_1) \dots (K_{\sigma_n} g_n)$ is related to Q', that is $P' \simeq_{\beta\eta} Q'$. We conclude that $K_\sigma P \simeq_{\beta\eta} H_\sigma Q$. ◀

We can now apply the parametricity Theorem 24 and obtain the following:

▶ **Theorem 27.** *Let M be a closed λ-term and σ a simple type. If $M \in |\sigma|$ then $M \in DIN_\sigma$.*

Proof. Let $M \in |\sigma|$ be a closed term and let $\sigma = \sigma_1 \to \dots \to \sigma_n \to Z_u$, for some $n \geq 0$. By propositions 25 and 26, $H_{\sigma_i} g_i \in |\sigma_i|_{\mathcal{M}_2}$, $K_{\sigma_i} g_i \in |\sigma_i|_{\mathcal{M}_1}$ and $(H_{\sigma_i} g_i)\ R^f[\sigma_i]\ (K_{\sigma_i} g_i)$. By Theorem 24, $M\ R^f[\sigma]\ M$, hence

$$M(K_{\sigma_1} g_1) \dots (K_{\sigma_n} g_n)\ R[Z_u]\ M(H_{\sigma_1} g_1) \dots (H_{\sigma_n} g_n) \tag{46}$$

that is $f_u\big(M(K_{\sigma_1} g_1) \dots (K_{\sigma_n} g_n)\big)\ \simeq_{\beta\eta}\ M(H_{\sigma_1} g_1) \dots (H_{\sigma_n} g_n)$, whence $M \in DIN_\sigma$. ◀

8 Conclusions

Two results were proved in this paper: first, that dinaturality implies (simple) typability (Theorem 14); second, that interpretability in the $\beta\eta$-stable semantics implies dinaturality (Theorem 27). By putting them together (along with Proposition 16) we obtain the following:

▶ **Theorem 28** (completeness of the $\beta\eta$-stable semantics)**.** *Let M be a closed λ-term and σ a simple type. If $M \in |\sigma|$, there exists $M' \simeq_{\beta\eta} M$ such that $\vdash M' : \sigma$.*

As we said, this completeness result can be proved by a direct argument ([6]) and extended to a restricted class of second order types ([3]). The core idea of the argument is the definition of a particular interpretation $\mathcal{M}$ such that $\mathcal{M}(Z_u)$ contains λ-terms which are $\beta\eta$-equivalent to terms which can be given type σ in an infinite context defined as Γ^∞ in the proof of Theorem 24.

This argument cannot be straightforwardly extended to the reducibility semantics commonly used to prove normalization theorems (for instance, Krivine's saturated families or Girard's reducibility candidates, see [4]). Indeed, the sets of λ-terms there considered have more complex closure properties, making in particular every term of the form $xP_1 \dots P_n$ belong to any considered set. Hence from the fact that a term belongs to the closure of the set of typable λ-terms one cannot deduce that the term is $\beta\eta$-equivalent to a typable term. In a word, the interpretation constructed over typable terms no longer captures typable terms only. Similarly, the proof of Theorem 24 cannot be straightforwardly extended to the reducibility semantics, as the interpretation $\mathcal{M}_P$ is defined starting from sets of terms which might well have the form $xP_1 \dots P_n$.

However, the results here presented suggest that completeness results might be looked for through a different path, namely that of showing that interpretable λ-terms satisfy parametricity and/or dinaturality, two semantic properties which allow, as it has been shown, to completely reconstruct the syntactic structure of $\beta\eta$-normal λ-terms.

References

1 E. S. Bainbridge, Peter J. Freyd, Andre Scedrov, and Philip J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70:35–64, 1990.

2 Joachim de Lataillade. Dinatural terms in System F. In *Proceedings of the Twenty-Fourth Annual IEEE Symposium on Logic in Computer Science (LICS 2009)*, pages 267–276, Los Angeles, California, USA, 2009. IEEE Computer Society Press.

3 Samir Farkh and Karim Nour. Résultats de complétude pour des classes de types du système *AF*2. *Informatique Théorique et Applications*, 31(6):513–537, 1998.

4 Jean Gallier. On Girard's "candidats de réductibilité". *Logic and Computer Science*, 1990.

5 Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Normal forms and cut-free proofs as natural transformations. In Y. Moschovakis, editor, *Logic from Computer Science*, volume 21, pages 217–241. Springer-Verlag, 1992.

6 Roger J. Hindley. The completeness theorem for typing λ-terms. *Theoretical Computer Science*, 22(1-2):1–17, 1983.

7 Jean-Louis Krivine. *Lambda calculus, types and models.* Ellis Horwood, 1993.

8 R. Labib-Sami. Types avec (ou sans) types auxiliaires. Manuscript, 1986.

9 Gordon Plotkin and Martin Abadi. A logic for parametric polymorphism. In *TLCA'93, International Conference on Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 361–375. Springer Berlin Heidelberg, 1993.

10 John C. Reynolds. Types, abstraction and parametric polymorphism. In R.E.A. Mason, editor, *Information Processing 1983*, pages 513–523. North-Holland, 1983.

A Postponed proofs

Proof of Theorem 5. For one direction first observe that the function `var` and `subt` are always defined: $\mathtt{var}(x)$ is the path π such that $x : \sigma_\pi$ occurs somewhere in the typing derivation of M; $\mathtt{subt}(\pi)$ is the path π' such that M_π has type $\sigma_{\pi'}$. So point 1 is satisfied. For the points 2 and 3 we argue by induction on the number of applications in M.

- if $M = \lambda x_{n+1}.\dots.\lambda x_{n+h}.x_j$, for $1 \leq j \leq n+h$, then $\mathcal{T}(N) = \{\epsilon\}$. Then $ar(\sigma_{\mathtt{var}\epsilon}) = ar(\sigma_j) = ar(\sigma) - (n+h) = ar(\sigma_{\mathtt{subt}(\epsilon)}) - ar(N_\epsilon)$, so point 2. holds and, moreover, $\sigma_{jl} = \sigma_{n+h+l}$ (as $\sigma_j = \tau = \sigma_{n+h+1} \to \dots \to \sigma_{n+h+ar(\sigma)} \to Z_u$) for all $1 \leq l \leq ar(\sigma) - (n+h) + 1$, so point 3. holds too.
- if $M = \lambda x_{n+1}.\dots.\lambda x_{n+h}.x_j M_1 \dots M_p$ where $1 \leq j \leq n+h$ and $p \geq 1$, then the typing derivation of M is as follows

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, \Delta \vdash x_j : \sigma_j \qquad \begin{matrix}\vdots \\ \Gamma, \Delta \vdash M_1 : \sigma_{j1}\end{matrix}}{\Gamma, \Delta \vdash x_j M_1 : \sigma_{j2} \to \dots \to \sigma_{jar(\sigma_j)} \to X}}{\begin{matrix}\ddots \\ \Gamma, \Delta \vdash x_j M_1 \dots M_{p-1} : \sigma_{j(p)} \to \dots \to \sigma_{jar(\sigma_j)} \to X\end{matrix}} \qquad \begin{matrix}\vdots \\ \Gamma, \Delta \vdash M_p : \sigma_{jp}\end{matrix}}{\Gamma, \Delta \vdash x_j M_1 \dots M_p : \sigma_{j(p+1)} \to \dots \to \sigma_{jar(\sigma_j)} \to X}}{\Gamma \vdash M : \tau} \tag{47}$$

where $\Delta = \{x_{n+1} : \sigma_{n+1}, \dots, x_{n+h} : \sigma_{n+h}\}$. Hence $\tau = \sigma_{n+1} \to \dots \to \sigma_{n+h} \to \sigma_{j(p+1)} \to \dots \to \sigma_{jar(\sigma_j)} \to Z_u$.
In order to apply the induction hypothesis to the terms $N_j = \lambda x_1.\dots.\lambda x_h.M_l$ and the types $\tau_j = \sigma_1 \to \dots \to \sigma_n \to \sigma_{ij}$, for $1 \leq l \leq p$, we must consider the maps $\mathtt{var}_j : Var(N_j) \to \mathcal{T}(\tau_j), \mathtt{subt}_j : \mathcal{T}(N_j) \to \mathcal{T}(\tau_j)$ and use equations 30, which are proved in detail below (end of the proof of Proposition 13). We verify now that any $\pi \in \mathcal{T}(N)$ verifies points 2 and 3 (by using equations 30):

1. if $\pi = \epsilon$, then $ar(\sigma_{\mathtt{var}(h(\epsilon))}) = ar(\sigma_j) = p+ar(\sigma)-(n+h) = p_N(\epsilon)+ar(\sigma_{\mathtt{subt}(\epsilon)})-ar(N_\epsilon)$. Otherwise $\pi = l * \pi'$, for some $1 \leq l \leq p$ and $\pi' \in \mathcal{T}(N_j)$; then $ar(\sigma_{\mathtt{var}(h(\pi))}) = ar(\sigma_{\mathtt{var}_j(h(\pi'))}) = p_{N_j}(\pi')+ar(\sigma_{\mathtt{subt}_j(\pi')})-ar(\big(N_j\big)_{\pi'}) = p_N(\pi)+ar(\sigma_{\mathtt{subt}(\pi)})-ar(N_\pi)$;
2. if $\pi = \epsilon$, then, for $1 \leq d \leq ar(\sigma) - ar(N) + 1$, $\sigma_{j(p+d)} = \sigma_{n+h+d} = \sigma_{ar(N)+d)}$ (since $\tau = \sigma_{n+1} \to \cdots \to \sigma_{n+h} \to \sigma_{j(p+1)} \to \cdots \to \sigma_{jar(\sigma_j)} \to Z_u$). Otherwise, $\pi = l*\pi'$ and, for $1 \leq d \leq ar(\sigma_{\mathtt{subt}(\pi)})-ar(N_\pi)+1$, $\sigma_{\mathtt{var}(h(\pi))*(p_N(\pi)+d)} = \sigma_{\mathtt{var}_j(h(\pi'))*(p_{N_j}(\pi')+d)} \overset{[i.h.]}{=} \sigma_{\mathtt{subt}_l(\pi')*(ar((N_j)_{\pi'})+d)} = \sigma_{\mathtt{subt}(\pi)*(ar(N_\pi)+d)}$.

For the converse direction, we argue again by induction on the number of applications in M:

- if $M = \lambda x_{n+1}.\ldots.\lambda x_{n+h}.x_j$, for $1 \leq j \leq n+h$, then from (1),(2) and (3), with $\pi = \epsilon$, it follows that $n+h \leq ar(\sigma)$, $ar(\sigma_j) = ar(\sigma)-(n+h)$ and, for $1 \leq l \leq ar(\sigma)-(n+h)+1 = ar(\tau)-h+1 = ar(\tau')+1$, where $\tau = \sigma_{n+1} \to \cdots \to \sigma_{n+h} \to \tau'$, $\sigma_{jl} = \sigma_{n+h+l}$. We deduce that $\sigma_j = \tau'$ and $\Gamma, \Delta \vdash x_j : \tau'$ is derivable (where $\Delta = \{x_{n+1} : \sigma_{n+1}, \ldots, x_{n+h} : \sigma_{n+h}\}$), hence $\Gamma \vdash M : \tau$ is derivable;
- if $M = \lambda x_{n+1}.\ldots.\lambda x_{n+h}.x_j M_1 \ldots M_p$, with $p \geq 1$, then we can apply the induction hypothesis to the terms $N_1, \ldots, N_p$ (defined as above), yielding $\Gamma, \Delta \vdash M_l : \sigma_{jl}$, with $\Delta = \{x_{n+1} : \sigma_{n+1}, \ldots, x_{n+h} : \sigma_{n+h}\}$, for all $1 \leq l \leq p$. From (1), (2) and (3), with $\pi = \epsilon$, it follows that $\tau = \sigma_{n+1} \to \cdots \to \sigma_{n+h} \to \tau'$, with $n+h \leq ar(\sigma)$, $ar(\sigma_j) = p+ar(\sigma)-(n+h)$ and, for $1 \leq l \leq ar(\sigma) - (n+h) + 1 = ar(\tau) - h + 1 = ar(\tau') + 1$, $\sigma_{j(p+l)} = \tau'_l$, hence $\sigma_j = \sigma_{j1} \to \cdots \to \sigma_{jp} \to \tau'$. So we can conclude, from $\Gamma, \Delta \vdash M_l : \sigma_{jl}$, for $1 \leq l \leq p$, that $\Gamma, \Delta \vdash x_j M_1 \ldots M_p : \tau'$ and finally that $\Gamma \vdash M : \tau$. ◀

End of the proof of Proposition 13. We will prove that, for all $\lambda \in \mathcal{T}(N_j)$, the equations 30 hold. For $\lambda \in \mathcal{T}(N_j)$, three cases arise for the variable x corresponding to $h_j(\lambda)$:

- x is one of the $x_1, \ldots, x_h$, i.e. $h_j(\lambda) = (\epsilon, l)$, for some $1 \leq l \leq h$;
- x is bound in M_j "at depth 0", i.e. $h_j(\lambda) = (\epsilon, h+l)$, for some $1 \leq l \leq ar(M_j)$;
- x it is bound in M_j "at depth > 0", i.e. $h_j(\lambda) = (\lambda' * k, l)$, for some λ' and $l \geq 1$.

We claim that:

1. in the first case, $\mathtt{var}(h(j * \lambda)) = \mathtt{var}_j(h_j(\lambda))$;
2. in the second case, $\mathtt{var}(h(j * \lambda)) = i * j * (\mathtt{var}_j(h_j(\lambda)) - h)$;
3. in the third case, two subcases must be considered:
   a. either $\mathtt{var}_j(h_j(\lambda)) = d * \lambda''$ with $d \leq h$, and $\mathtt{var}(h(j * \lambda)) = \mathtt{var}_j(h_j(\lambda))$;
   b. or $\mathtt{var}_j(h_j(\lambda)) = (h+d) * \lambda''$ with $d \leq ar(M_j)$, and $\mathtt{var}(h(j * \lambda)) = i * j * d * \lambda''$.

We prove now our claims:

1. $\mathtt{var}(h(j * \lambda)) = \mathtt{var}(\epsilon, l) = l = \mathtt{var}_j(\epsilon, l) = \mathtt{var}_j(h_j(\epsilon))$;
2. $\mathtt{var}(h(j*\lambda)) = \mathtt{var}(j, l) = \mathtt{var}(h(\epsilon)) * j * l = i * j * l = i * j * (l+h-h) = i * j * (\mathtt{var}_j(\epsilon, l+h) - h) = i * j * (\mathtt{var}_j(h_j(\lambda)) - h)$;
3. we consider the two subcases separately:
   a. if $\mathtt{var}_j(h_j(\lambda)) = d * \lambda''$ with $d \leq h$, then we argue by induction on $\ell(\lambda)$: we have $\mathtt{var}(h(j * \lambda' * k)) = \mathtt{var}(j * \lambda' * k, l) = \mathtt{var}(h(j * \lambda')) * k * l \overset{*}{=} \mathtt{var}_j(h_j(\lambda')) * k * l = \mathtt{var}_j(\lambda' * k, l) = \mathtt{var}_j(h_j(\lambda))$ where in the starred passage, if $\lambda' = \epsilon$, we apply point 1. as $var_j(h_j(\lambda')) = var_j(\epsilon, d) = d$ (since $\epsilon \notin Im(var_j)$ and $\ell(h_j(\epsilon)_1) \leq 0$), while if $\lambda' \neq \epsilon$, we apply the induction hypothesis to λ' as $\ell(\lambda') < \ell(\lambda)$ (since $\ell(h_j(\lambda)_1) \leq \ell(\lambda)$) and $h_j(\lambda') = d * o$ for some path o;
   b. if $\mathtt{var}_j(h_j(\lambda)) = (h+d) * \lambda''$ with $d \leq h$, then again we argue by induction on $\ell(\lambda)$: we have $\mathtt{var}(h(j * \lambda)) = \mathtt{var}(j * \lambda' * k, l) = \mathtt{var}(h(j * \lambda')) * k * l$. If $\lambda' = \epsilon$ then, by point 2. we have $\mathtt{var}(h(j * \lambda')) = i * j * (\mathtt{var}_j(h_j(\lambda')) - h)$, hence $\mathtt{var}(h(j * \lambda')) * k * l =$

$i*j*(\mathtt{var}_j(h_j(\lambda'))-h)*k*l = i*j*d*k*l = i*j*d*\lambda''$, where $k*l=\lambda''$ follows from $\mathtt{var}_j(h_j(\lambda)) = \mathtt{var}_j(\lambda'*k,l) = \mathtt{var}_j(h_j(\lambda')*k*l) = (h+d)*k*l = (h+d)*\lambda''$. If $\lambda' \neq \epsilon$ then we can apply the induction hypothesis to λ' as $\ell(\lambda') < \ell(\lambda)$ and $\mathtt{var}_j(h_j(\lambda')) = (h+d)*o$, for some path o; so we get that $\mathtt{var}(h(j*\lambda')) = i*j*d*o$ and then we can compute $\mathtt{var}(h(j*\lambda)) = \mathtt{var}(j*\lambda'*k,l) = \mathtt{var}(h(j*\lambda'))*k*l = i*j*d*o*k*l = i*j*d*\lambda''$, where $o*k*l=\lambda''$ follows from $\mathtt{var}_j(h_j(\lambda)) = \mathtt{var}_j(\lambda'*k,l) = \mathtt{var}_j(h_j(\lambda')*k*l) = (h+d)*o*k*l = (h+d)*\lambda''$.

We can now verify that $(\tau_j)_{\mathtt{var}_j(h_j(\pi'))} = \sigma_{\mathtt{var}(h(\pi))}$, for $\pi = j*\pi'$. We must consider the tree cases above:

1. if $h_j(\pi') = (\epsilon, l)$ with $l \leq h$, then $(\tau_j)_{\mathtt{var}_j(h_j(\pi))} = (\tau_j)_{\mathtt{var}_j(\epsilon,l)} = (\tau_j)_l = \sigma_l$ and $\sigma_{\mathtt{var}(h(j*\pi''))} = \sigma_{\mathtt{var}_j(h_j(\pi'))} = \sigma_{\mathtt{var}_j(\epsilon,l)} = \sigma_l$;
2. if $h_j(\pi') = (\epsilon, h+l)$, then $(\tau_j)_{\mathtt{var}_j(h_j(\pi'))} = (\tau_j)_{(h+l)} = \sigma_{i*j*l} = \sigma_{i*j*(\mathtt{var}_j(h_j(\pi'))-h)} = \sigma_{\mathtt{var}(h(j*\pi'))} = \sigma_{\mathtt{var}(h(\pi))}$;
3. if $h_j(\pi') = (\lambda'*r,l)$, then we consider the two subcases:
   a. if $\mathtt{var}_j(h_j(\pi')) = d*\lambda$ with $d \leq h$, then $(\tau_j)_{\mathtt{var}_j(h_j(\pi'))} = (\tau_j)_{d*\lambda} = \sigma_{d*\lambda} = \sigma_{\mathtt{var}_j(h_j(\pi'))} = \sigma_{\mathtt{var}(h(j*\pi'))} = \sigma_{\mathtt{var}(h(\pi))}$;
   b. if $\mathtt{var}_j(h_j(\pi')) = (h+d)*\lambda$ with $d \leq ar(M_j)$, then $(\tau_j)_{\mathtt{var}_j(h_j(\pi'))} = (\tau_j)_{(h+d)*\lambda'} = \sigma_{i*j*d*\lambda'} = \sigma_{\mathtt{var}(h(j*\pi'))} = \sigma_{\mathtt{var}(h(\pi))}$.

Finally, for $\pi' = \pi''*k$, we have that $(\tau_j)_{\mathtt{subt}_j(\pi')} = (\tau_j)_{\mathtt{var}_j(h_j(\pi''))*k} = \sigma_{\mathtt{var}(h(j*\pi''))*k} = \sigma_{\mathtt{subt}(\pi)}$. ◀

A Curry-Howard Approach to Church's Synthesis*

Pierre Pradic[1] and Colin Riba[2]

1 **ENS de Lyon, Université de Lyon, LIP†, Lyon, France; and University of Warsaw, Faculty of Mathematics, Informatics and Mechanics, Warsaw, Poland**
`pierre.pradic@ens-lyon.fr`

2 **ENS de Lyon, Université de Lyon, LIP‡, Lyon, France**
`colin.riba@ens-lyon.fr`

Abstract

Church's synthesis problem asks whether there exists a finite-state stream transducer satisfying a given input-output specification. For specifications written in Monadic Second-Order Logic over infinite words, Church's synthesis can theoretically be solved algorithmically using automata and games. We revisit Church's synthesis *via* the Curry-Howard correspondence by introducing SMSO, a non-classical subsystem of MSO, which is shown to be sound and complete w.r.t. synthesis thanks to an automata-based realizability model.

1998 ACM Subject Classification F.4.1 Mathematical Logic.

Keywords and phrases Intuitionistic Arithmetic, Realizability, Monadic Second-Order Logic on Infinite Words

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.30

1 Introduction

Church's synthesis [5] consists in the automatic extraction of stream transducers (or *Mealy machines*) from input-output specifications, typically written in some subsystem of *Monadic Second-Order Logic* (MSO) over ω-words. MSO over ω-words is a decidable logic by Büchi's Theorem [3]. It subsumes non-trivial logics used in verification such as LTL (see e.g. [16, 10]).

Traditional approaches to synthesis (see e.g. [17]) are based, *via* McNaughton's Theorem [9], on the translation of MSO-formulae to *deterministic* automata on ω-words (such as *Muller* or *parity* automata)[1]. Such automata are then turned into game graphs, in which the *Opponent* O (∀bélard) plays input characters to which the *Proponent* P (∃loïse) replies with output characters. Solutions to Church's synthesis are then given by the Büchi-Landweber Theorem [4], which says that in such games, either P or O has finite-state winning strategy.

Fully automatic approaches to synthesis suffer from prohibitively high computational costs, essentially for the following two reasons. First, the translation of MSO-formulae to automata is non-elementary, and McNaughton Theorem involves a non-trivial powerset construction (such as *Safra construction*, see e.g. [16, 10]). Second, similarly as with other automatic verification techniques based on Model Checking, the solution of parity games ultimately relies on exhaustive state exploration. While they have had (and still have)

* This work was partially supported by the ANR-14-CE25-0007 – RAPIDO and the ANR-BLANC-SIMI-2-2011 – RECRÉ.

† UMR 5668 CNRS ENS Lyon UCBL INRIA.

‡ UMR 5668 CNRS ENS Lyon UCBL INRIA.

[1] A solution is also possible *via* tree automata [11].

considerable success for verifying concurrency properties, such techniques hardly managed up to now to give practical algorithms for synthesis (even for fragments of LTL, see e.g. [1]).

In this work, we propose a Curry-Howard approach to Church's synthesis based on a proof system allowing for human intervention and compositional reasoning. In a typical usage scenario, the user interactively performs some proofs steps and delegate the generated subgoals to automatized synthesis procedures. The partial proof tree built by the user is then translated to a combinator able to compose the transducers synthesized by the automatic procedures[2]. Having in mind that interactive proof systems (such as Coq [15]) have known in the last decade an explosion of large developments, we believe that semi-automatic approaches like ours could ultimately help mitigate the algorithmic costs of synthesis, in particular in helping to combine automatic methods with human intervention.

The Curry-Howard correspondence asserts that, given a suitable proof system, any proof therein can be interpreted as a program. Actually, *via* the Curry-Howard correspondence, the soundness of many type/proof systems is proved by means of *realizability*, which tells how to read a formula from the logic as a specification for a program. Our starting point is the fact that MSO on ω-words can be completely axiomatized as a subsystem of second-order Peano arithmetic [14] (see also [12]). From the classical axiomatization of MSO, we derive an intuitionistic system SMSO equipped with an extraction procedure which is sound and complete w.r.t. Church's synthesis: proofs of existential statements can be translated to Mealy machines and such proofs exist for all solvable instances of Church's synthesis. The key point in our approach is that on the one hand, finite-state realizers[3] are constructively extracted from proofs in SMSO, while on the other hand, their correctness involves the full power of MSO. So in particular, our adaptation of the usual Adequacy Lemma of realizability does rely on the non-constructive proof of correctness of deterministic automata obtained by McNaughton's Theorem (see e.g. [16]), while these automata do not have to be concretely built during the extraction procedure.

The paper is organized as follows. We first recall in §2 some background on MSO and Church's synthesis. Our intuitionistic system SMSO is then presented in §3. Section 4 provides some technical material as well as detailed examples on the representation of Mealy machines in MSO, and §5 presents our realizability model.

2 Church's Synthesis and MSO on Infinite Words

Notations. Alphabets (denoted Σ, Γ, etc) are finite non-empty sets. Concatenation of words s, t is denoted either $s.t$ or $s \cdot t$, and ε is the empty word. We use the vectorial notation both for words and finite sequences, so that e.g. $\overline{B}$ denotes a finite sequence $B_1, \ldots, B_n$ and $\overline{\mathsf{a}}$ denotes a word $\mathsf{a}_1.\cdots.\mathsf{a}_n \in \Sigma^*$. Given an ω-word (or stream) $B \in \Sigma^\omega$ and $n \in \mathbb{N}$ we write $B{\restriction}n$ for the finite word $B(0).\cdots.B(n-1) \in \Sigma^*$. For each $k \in \mathbb{N}$, we still write k for the function from $\mathbb{N}$ to $\mathbf{2}$ which takes n to 1 iff $n = k$.

---

[2] We thank the anonymous referee who urged us to state this explicitly.

[3] We use the word *realizer* with two historically distinct meanings. In the context of Church's synthesis, a realizer of a $\forall\exists$-formula is a transducer which witnesses the $\forall\exists$ by computing an instantiation of the existential variables while reading input values for the universal variables (see e.g. [1]). In (constructive) proof theory, *realizability* is a relation between programs (the realizers) and formulae, usually defined by induction on formulae (see e.g. [8]). A realizer of a $\forall\exists$-formula consists of a function witnessing the $\forall\exists$, together with a realizer witnessing the correctness of that function.

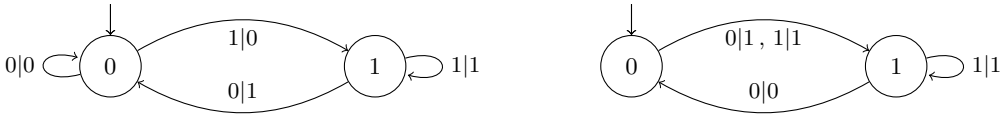


Figure 1 Examples of Mealy Machines (where a transition $\mathtt{a}|\mathtt{b}$ outputs $\mathtt{b}$ from input $\mathtt{a}$).

Church's Synthesis and Synchronous Functions. Church's synthesis consists in the automatic extraction of stream transducers (or *Mealy machines*) from input-output specifications (see e.g. [17]). As a typical specification, consider, for a machine which outputs streams $B \in \mathbf{2}^\omega$ from input streams $A \in \mathbf{2}^\omega$, the behavior (from [17]) expressed by

$$\Phi(A,B) \quad \overset{\text{def.}}{\iff} \quad \begin{cases} \forall n (A(n) = 1 \implies B(n) = 1) & \text{and} \\ \forall n (B(n) = 0 \implies B(n+1) = 1) & \text{and} \\ (\exists^\infty n\ A(n) = 0) \implies (\exists^\infty n\ B(n) = 0) \end{cases} \tag{1}$$

In words, the relation $\Phi(A,B)$ imposes $B(n) \in \mathbf{2}$ to be 1 whenever $A(n) \in \mathbf{2}$ is 1, B not to be 0 in two consecutive positions, and moreover B to be infinitely often 0 whenever A is infinitely often 0. We are interested in the realization of such specifications by finite-state stream transducers or *Mealy machines*.

▶ **Definition 2.1** (Mealy Machine)**.** A *Mealy machine* $\mathcal{M}$ with input alphabet Σ and output alphabet Γ (notation $\mathcal{M} : \Sigma \to \Gamma$) is given by a finite set of states Q with a distinguished initial state $q^\iota \in Q$, and a transition function $\partial : Q \times \Sigma \to Q \times \Gamma$.

We often write ∂^o for $\pi_2 \circ \partial : Q \times \Sigma \to \Gamma$ and ∂^* for the map $\Sigma^* \to Q$ obtained by iterating ∂ from the initial state: $\partial^*(\varepsilon) := q^\iota$ and $\partial^*(\overline{\mathtt{a}}.\mathtt{a}) := \pi_1(\partial(\partial^*(\overline{\mathtt{a}}), \mathtt{a}))$

A Mealy machine $\mathcal{M} : \Sigma \to \Gamma$ induces a function $F : \Sigma^\omega \to \Gamma^\omega$ obtained by iterating ∂^o along the input: $F(B)(n) = \partial^o(\partial^*(B\restriction n), B(n))$. Hence F can produce a length-n prefix of its output from a length-n prefix of its input. These functions are called *synchronous*.

▶ **Definition 2.2** (Synchronous Function)**.** A function $F : \Sigma^\omega \to \Gamma^\omega$ is *synchronous* if for all $n \in \mathbb{N}$ and all $A, B \in \Sigma^\omega$ we have $F(A)\restriction n = F(B)\restriction n$ whenever $A\restriction n = B\restriction n$. We say that a synchronous function F is *finite-state* if it is induced by a Mealy machine.

▶ **Example 2.3.**

(a) The identity function $\Sigma^\omega \to \Sigma^\omega$ is induced by the Mealy machine with state set $\mathbf{1} = \{\bullet\}$ and identity transition function $\partial : (\bullet, \mathtt{a}) \longmapsto (\bullet, \mathtt{a})$.

(b) The Mealy machine depicted on Fig. 1 (left) induces a synchronous function $F : \mathbf{2}^\omega \to \mathbf{2}^\omega$ such that $F(B)(n+1) = 1$ iff $B(n) = 1$.

(c) The Mealy machine depicted on Fig. 1 (right), taken from [17], induces a synchronous function which realizes the specification (1).

(d) Synchronous functions are obviously continuous (taking the product topology on Σ^ω and Γ^ω, with Σ, Γ discrete), but there are continuous functions which are not synchronous, for instance the function $P : \mathbf{2}^\omega \to \mathbf{2}^\omega$ such that $P(A)(n) = 1$ iff $A(n+1) = 1$.

For the definition and adequacy of our realizability interpretation, it turns out to be convenient to work with a category of finite-state synchronous functions.

▶ **Definition 2.4.** Let $\mathbf{M}$ be the category whose objects are alphabets and whose maps from Σ to Γ are finite-state synchronous functions $F : \Sigma^\omega \to \Gamma^\omega$.

$$\begin{array}{rrcl} \textbf{Atoms:} & \alpha & ::= & x \doteq y \mid x \dot{\leq} y \mid \mathsf{S}(x,y) \mid \mathsf{Z}(x) \mid x \mathrel{\dot{\in}} X \mid \top \mid \bot \\ \textbf{Deterministic formulae:} & \delta, \delta' & ::= & \alpha \mid \delta \wedge \delta' \mid \neg\varphi \\ \textbf{MSO formulae:} & \varphi, \psi & ::= & \delta \mid \varphi \wedge \psi \mid \exists x\, \varphi \mid \exists X\, \varphi \end{array}$$

Figure 2 The Formulae of MSO.

Note that functions $\mathtt{f} : \Sigma \to \Gamma$ induce $\mathbf{M}$-maps $[\mathtt{f}] : \Sigma \to_{\mathbf{M}} \Gamma$. Also, $\mathbf{M}$ has finite products.

▶ **Proposition 2.5.** *The category* $\mathbf{M}$ *has finite products. The product of* $\Sigma_1, \ldots, \Sigma_n$ *(for* $n \geq 0$*) is given by the* $\mathbf{Set}$*-product* $\Sigma_1 \times \cdots \times \Sigma_n$ *(so that* $\mathbf{1}$ *is terminal in* $\mathbf{M}$*).*

Monadic Second-Order Logic (MSO) on Infinite Words. We consider a formulation of MSO based on a purely relational two-sorted language, with a specific choice of atomic formulae. There is a sort of *individuals*, with variables x, y, z, etc, and a sort of *(monadic) predicates*, with variables X, Y, Z, etc. Our formulae for MSO, denoted φ, ψ, etc are given on Fig. 2. They are defined by mutual induction with the *deterministic formulae* (denoted δ, δ', etc) from atomic formulae ranged over by α.

MSO formulae are interpreted in the standard model $\mathfrak{N}$ of ω-words as usual. Individual variables range over natural numbers $n, m, \ldots \in \mathbb{N}$ and predicate variables range over sets of natural numbers $A, B, \ldots \in \mathcal{P}(\mathbb{N}) \simeq \mathbf{2}^{\omega}$. The atomic predicates are interpreted as expected: $\doteq$ is equality, $\dot{\in}$ is membership, $\dot{\leq}$ is the relation $\leq$ on $\mathbb{N}$, S is the successor relation, and Z holds on n iff $n = 0$. We often write $X(x)$ or even Xx for $x \mathrel{\dot{\in}} X$. As usual we let:

$$\varphi \to \psi \;:=\; \neg(\varphi \wedge \neg\psi) \qquad \varphi \vee \psi \;:=\; \neg(\neg\varphi \wedge \neg\psi) \qquad \forall(-)\, \varphi \;:=\; \neg\exists(-)\, \neg\varphi$$

MSO on ω-words is known to be decidable by Büchi's Theorem [3].

▶ **Theorem 2.6** (Büchi [3])**.** MSO *over* $\mathfrak{N}$ *is decidable.*

Following [3] (but see also e.g. [10]), the (non-deterministic) automata method for deciding MSO proceeds by a recursive translation of MSO-formulae to *Büchi automata*. A *Büchi automaton* is a non-deterministic finite state automaton running on ω-words. Büchi automata are equipped with a set of final states, and a run on an ω-word is accepting if it has infinitely many occurrences of final states.

The crux of Büchi's Theorem is the effective closure of Büchi automata under complement. Let us recall a few known facts (see e.g. [16, 7]). First, the translation of MSO-formulae to automata is non-elementary. Second, its is known that *deterministic* Büchi automata are strictly less expressive than non-deterministic ones. Finally, it is known that complementation of Büchi automata is algorithmically hard: there is a family of languages $(\mathcal{L}_n)_{n>0}$ such that each $\mathcal{L}_n$ can be recognized by a Büchi automaton with $n+2$ states, but such that the complement of $\mathcal{L}_n$ can not be recognized by a Büchi automaton with less than $n!$ states.

Church's Synthesis for MSO. Church's synthesis problem for MSO is the following. Given as input an MSO formula $\varphi(\overline{X}; \overline{Y})$ (where $\overline{X} = X_1, \ldots, X_q$ and $\overline{Y} = Y_1, \ldots, Y_p$), (1) decide whether there exist finite-state synchronous functions $\overline{F} = F_1, \ldots, F_p : \mathbf{2}^q \to_{\mathbf{M}} \mathbf{2}$ such that $\mathfrak{N} \models \varphi(\overline{A}; \overline{F}(\overline{A}))$ for all $\overline{A} \in (\mathbf{2}^{\omega})^q \simeq (\mathbf{2}^q)^{\omega}$, and (2), construct such $\overline{F}$ whenever they exist.

▶ **Example 2.7.** The specification Φ displayed in (1) can be officially written in the language of MSO as the following formula $\phi(X;Y)$ (where $\exists^\infty t\ \varphi(t)$ stands for $\forall x \exists t(t \dot{\geq} x \wedge \varphi(t))$):

$$\phi(X;Y) \ := \ \forall t\,(Xt \to Yt) \wedge \forall t,t'\,(\mathsf{S}(t,t') \to \neg Yt \to Yt') \wedge [(\exists^\infty t\,\neg Xt) \to (\exists^\infty t\,\neg Yt)]$$

Church's synthesis has been solved by Büchi & Landweber [4], using automata on ω-words and infinite two-player games (a solution is also possible *via* tree automata [11]): there is an algorithm which, on input $\varphi(\overline{X};\overline{Y})$, (1) decides when a synchronous realizer of $\varphi(\overline{X};\overline{Y})$ exists, (2) provides a finite-state Mealy machine implementing it[4], and (3) moreover provides a synchronous finite-state counter realizer (*i.e.* a realizer of $\psi(\overline{Y};\overline{X}) := \neg\varphi(\overline{X};\overline{Y})$) when no synchronous realizer of $\varphi(\overline{X};\overline{Y})$ exists.

The standard algorithm solving Church's synthesis for MSO (see e.g. [17]) proceeds *via* McNaughton's Theorem ([9], see also e.g. [10, 16]), which states that Büchi automata can be translated to equivalent *deterministic* finite state automata, but equipped with stronger acceptance conditions than Büchi automata. There are different variants of such conditions (*Muller*, *Rabin*, *Streett* or *parity* conditions, see e.g. [16, 7]). All of them allow to specify which states an infinite run *must not* see infinitely often. For the purpose of this paper, we only need to consider the simplest of them, the Muller conditions. A *Muller condition* is given by a family of set of states $\mathcal{T}$, and a run is accepting when the set of states occurring infinitely often in it belongs to the family $\mathcal{T}$.

▶ **Theorem 2.8** (McNaughton [9])**.** *Each Büchi automaton is equivalent to a deterministic Muller automaton.*

There is a lower bound in $2^{O(n)}$ for the number of states of a Muller automaton equivalent to a Büchi automaton with n states. The best known constructions for McNaughton's Theorem (such as *Safra's construction* or its variants) give deterministic Muller automata with $2^{O(n\log(n))}$ states from non-deterministic Büchi automata with n states.

The standard solution to Church's synthesis for MSO starts by translating $\varphi(\overline{X};\overline{Y})$ to a deterministic Muller automaton, and then turns this deterministic automaton into a two-player sequential game, in which the Opponent ∀bélard plays inputs bit sequences in $\mathbf{2}^p$ while the Proponent ∃loïse replies with outputs bit sequences in $\mathbf{2}^q$. The game is equipped with an ω-regular winning condition (induced by the acceptance condition of the Muller automaton). The solution is then provided by Büchi-Landweber's Theorem, which states that ω-regular games on finite graphs are effectively determined, and moreover that the winner always has a finite state winning strategy.

▶ **Example 2.9.** Consider the last conjunct $\phi_2[X,Y] := (\exists^\infty t\ \neg Xt) \to (\exists^\infty t\ \neg Yt)$ of the formula $\phi(X;Y)$ of Ex. 2.7. When translating ϕ_2 to a finite state automaton, the positive occurrence of $(\exists^\infty t\ \neg Yt)$ can be translated to a deterministic Büchi automaton. However, the negative occurrence of $(\exists^\infty t\ \neg Xt)$ corresponds to $(\forall^\infty t\ Xt) = (\exists n\,\forall t \dot{\geq} n\,Xt)$ and can not be translated to a *deterministic* Büchi automaton. Even if a very simple two-state Muller automaton exists for $(\forall^\infty t\ Xt)$, McNaughton's Theorem 2.8 is in general required for positive occurrences of the form $\forall^\infty t\ (-)$.

An Axiomatization of MSO. Our approach to Church's synthesis relies on the fact that the MSO-theory of $\mathfrak{N}$ can be completely axiomatized as a subsystem of second-order Peano

[4] It follows from the finite-state determinacy of ω-regular games that a finite-state synchronous realizer exists whenever a synchronous realizer exists (see e.g. [17]).

$$\frac{}{\overline{\varphi} \vdash t \doteq t} \qquad \frac{\overline{\varphi} \vdash \varphi[t/z] \qquad \overline{\varphi} \vdash t \doteq u}{\overline{\varphi} \vdash \varphi[u/z]} \qquad \frac{\overline{\varphi} \vdash x \dot{\leq} y \qquad \overline{\varphi} \vdash y \dot{\leq} x}{\overline{\varphi} \vdash x \doteq y}$$

$$\frac{}{\overline{\varphi} \vdash x \dot{\leq} x} \qquad \frac{\overline{\varphi} \vdash \mathsf{S}(x,y)}{\overline{\varphi} \vdash x \dot{\leq} y} \qquad \frac{\overline{\varphi} \vdash x \dot{\leq} y \qquad \overline{\varphi} \vdash y \dot{\leq} z}{\overline{\varphi} \vdash x \dot{\leq} z} \qquad \frac{}{\overline{\varphi}, \mathsf{S}(x,y), \mathsf{Z}(y) \vdash \bot}$$

$$\frac{}{\overline{\varphi} \vdash \exists y\, \mathsf{Z}(y)} \qquad \frac{}{\overline{\varphi} \vdash \exists y\, \mathsf{S}(x,y)} \qquad \frac{}{\overline{\varphi}, \mathsf{S}(y,y'), x \dot{\leq} y', \neg(x \doteq y') \vdash x \dot{\leq} y}$$

$$\frac{}{\overline{\varphi}, \mathsf{S}(y,x), \mathsf{S}(z,x) \vdash y \doteq z} \qquad \frac{}{\overline{\varphi}, \mathsf{Z}(x), \mathsf{Z}(y) \vdash x \doteq y} \qquad \frac{}{\overline{\varphi}, \mathsf{S}(x,y), \mathsf{S}(x,z) \vdash y \doteq z}$$

Figure 3 Arithmetic Rules of MSO and SMSO.

arithmetic [14] (see also [12]). We consider a specific set of axioms which consists of the rules depicted on Fig. 3 together with the following *comprehension* and *induction* rules

$$\frac{\overline{\varphi} \vdash \varphi[\psi[y]/X]}{\overline{\varphi} \vdash \exists X\, \varphi} \qquad \frac{\overline{\varphi}, \mathsf{Z}(z) \vdash \varphi[z/x] \qquad \overline{\varphi}, \mathsf{S}(y,z), \varphi[y/x] \vdash \varphi[z/x]}{\overline{\varphi} \vdash \varphi} \tag{2}$$

where z and y do not occur free in $\overline{\varphi}, \varphi$, and where $\varphi[\psi[y]/X]$ is the usual formula substitution, which commutes over all connectives (avoiding the capture of free variables), and with $(x \mathrel{\dot{\in}} X)[\psi[y]/X] \;=\; \psi[x/y]$.

▶ **Theorem 2.10** ([14]). *For every (closed)* MSO*-formula φ, we have $\mathfrak{N} \models \varphi$ if and only if $\vdash \varphi$ is derivable in classical two-sorted predicate logic with the rules of Fig. 3 and (2).*

3 A Synchronous Intuitionistic Restriction of MSO

We now introduce SMSO, an intuitionistic restriction of MSO. As expected, SMSO contains MSO *via* negative translation. But thanks to its vocabulary without primitive universals, SMSO actually admits a Glivenko Theorem, so that SMSO proves $\neg\neg\varphi$ whenever MSO $\vdash \varphi$. Moreover, SMSO is equipped with an extraction procedure which is sound and complete w.r.t. Church's synthesis: proofs of existential statements can be translated to finite state synchronous realizers, and such proofs exist for all solvable instances of Church's synthesis.

As it is common with intuitionistic versions of classical systems, SMSO has the same language as MSO, and its deduction rules are based on intuitionistic predicate calculus. Moreover, since (monadic) predicate variables are computational objects in our realizability interpretation, similarly as with higher-type Heyting arithmetic (see e.g. [8]), SMSO has a comprehension scheme which corresponds to the negative translation of the full comprehension scheme of MSO[5]. On the other hand, for the extraction of *synchronous* realizers from proofs, SMSO has a restricted induction scheme corresponding to the negative translation of the induction scheme of MSO. As a consequence, and in contrast with usual versions of intuitionistic (Heyting) arithmetic, this restricted induction scheme is not able to prove the elimination of double negation on atomic formulae. Fortunately, all atomic formulae of MSO can be interpreted by *deterministic Büchi* automata, and have a trivial computational

[5] In contrast with Girard's System F [6], in which second-order variables have no computational content.

$$\frac{}{\overline{\varphi}, \varphi \vdash \varphi} \qquad \frac{\overline{\varphi} \vdash \psi \qquad \overline{\varphi}, \psi \vdash \varphi}{\overline{\varphi} \vdash \varphi} \qquad \frac{}{\overline{\varphi}, \neg\neg\delta \vdash \delta} \qquad \frac{\overline{\varphi} \vdash \varphi \qquad \overline{\varphi} \vdash \neg\varphi}{\overline{\varphi} \vdash \bot} \qquad \frac{\overline{\varphi} \vdash \bot}{\overline{\varphi} \vdash \varphi}$$

$$\frac{\overline{\varphi} \vdash \varphi \qquad \overline{\varphi} \vdash \psi}{\overline{\varphi} \vdash \varphi \wedge \psi} \qquad \frac{\overline{\varphi} \vdash \varphi \wedge \psi}{\overline{\varphi} \vdash \varphi} \qquad \frac{\overline{\varphi} \vdash \varphi \wedge \psi}{\overline{\varphi} \vdash \psi} \qquad \frac{\overline{\varphi} \vdash \varphi[y/x]}{\overline{\varphi} \vdash \exists x\, \varphi} \qquad \frac{\overline{\varphi} \vdash \varphi[Y/X]}{\overline{\varphi} \vdash \exists X\, \varphi}$$

$$\frac{\overline{\varphi}, \varphi \vdash \psi \qquad \overline{\varphi} \vdash \exists x\, \varphi}{\overline{\varphi} \vdash \psi}\ (x \text{ not free in } \overline{\varphi}, \psi) \qquad \frac{\overline{\varphi}, \varphi \vdash \psi \qquad \overline{\varphi} \vdash \exists X\, \varphi}{\overline{\varphi} \vdash \psi}\ (X \text{ not free in } \overline{\varphi}, \psi)$$

Figure 4 Logical Rules of SMSO (where δ is deterministic).

content. This more generally leads to the notion of *deterministic* formulae (see Fig. 2), which contain negative formulae and atomic formulae. Deterministic formulae will be interpreted by deterministic (not nec. Büchi) automata, and have trivial realizers. We can therefore have as axiom the elimination of double negation for deterministic formulae, which are thus the SMSO counterpart of the formulae of Heyting arithmetic admitting elimination of double negation (see e.g. [8]).

Furthermore, SMSO is equipped with a positive *synchronous* restriction of comprehension, which allows to have realizers for all solvable instances of Church's synthesis. The synchronous restriction of comprehension asks the comprehension formula to be *uniformly bounded* in the following sense.

▶ **Definition 3.1.**

(i) Given MSO-formulae φ and θ and a variable y, the *relativization of φ to $\theta[y]$* (notation $\varphi\restriction\theta[y]$), is defined by induction on φ as usual:

$$\alpha\restriction\theta[y] := \alpha \qquad (\varphi \wedge \psi)\restriction\theta[y] := \varphi\restriction\theta[y] \wedge \psi\restriction\theta[y] \qquad (\neg\varphi)\restriction\theta[y] := \neg\varphi\restriction\theta[y]$$

$$(\exists X\, \varphi)\restriction\theta[y] := \exists X\, \varphi\restriction\theta[y] \qquad (\exists x\, \varphi)\restriction\theta[y] := \exists x\, (\theta[x/y] \wedge \varphi\restriction\theta[y])$$

where, in the clauses for $\exists$, the variables x and X are assumed not to occur free in θ. Note that y does not occur free in $\varphi\restriction\theta[y]$.

(ii) An MSO-formula $\hat{\varphi}$ is *bounded by x* if it is of the form $\psi\restriction(y \dot{\leq} x)[y]$ (notation $\psi\restriction[- \dot{\leq} x]$). It is *uniformly bounded* if moreover x is the only free individual variable of $\hat{\varphi}$.

As we shall see in §4.3, bounded formulae are exactly those definable in MSO over finite words. We are now ready to define the system SMSO.

▶ **Definition 3.2** (SMSO)**.** The logic SMSO has the same language as MSO. Its deduction rules are those given in Fig. 4 together with the rules of Fig. 3 and with the following rules of resp. *negative comprehension*, *deterministic induction* (where x and y do not occur free in $\overline{\varphi}, \delta$) and *synchronous comprehension* in which $\hat{\varphi}$ is uniformly bounded by y:

$$\frac{\overline{\varphi} \vdash \psi[\varphi[y]/X]}{\overline{\varphi} \vdash \neg\neg\exists X\, \psi} \qquad \frac{\overline{\varphi}, \mathsf{Z}(z) \vdash \delta[z/x] \qquad \overline{\varphi}, \mathsf{S}(y, z), \delta[y/x] \vdash \delta[z/x]}{\overline{\varphi} \vdash \delta} \qquad \frac{\overline{\varphi} \vdash \psi[\hat{\varphi}[y]/X]}{\overline{\varphi} \vdash \exists X\, \psi}$$

▶ Remark. The axiom $\overline{\varphi}, \neg\neg\delta \vdash \delta$ of double negation elimination for deterministic formulae would already be derivable in a version of SMSO where this axiom is weakened to double negation elimination for atomic formulae. We take $\overline{\varphi}, \neg\neg\delta \vdash \delta$ as an axiom because it admits trivial realizers. Similarly, the cut rule is admissible, but we include it since we have a direct composition of realizers.

A Glivenko Theorem for SMSO. Thanks to its limited vocabulary, SMSO satisfies a Glivenko theorem, and thus a very simple negative translation from MSO. Glivenko's theorem is usually stated only for propositional logic, but can be extended to formulae containing existentials; the impossible case is the universal quantification. In particular, should one extend the logical constructs with universal quantification by freely adjoining them to SMSO, this would no longer hold. This would actually not be such a severe consequence since our results would also hold with a usual recursive negative translation instead of $\neg\neg(-)$.

▶ **Theorem 3.3.** *If* $\mathsf{MSO} \vdash \varphi$*, then* $\mathsf{SMSO} \vdash \neg\neg\varphi$*.*

The Main Result. We are now ready to state the main result of this paper, which says that SMSO is correct and complete (w.r.t. its provable existentials) for Church's synthesis.

▶ **Theorem 3.4** (Main Theorem)**.** *Consider an* MSO*-formula* $\varphi(\overline{X};\overline{Y})$*.*

(i) *From a proof of* $\exists\overline{Y}\,\neg\neg\varphi(\overline{X};\overline{Y})$ *in* SMSO*, one can extract a finite-state synchronous realizer of* $\varphi(\overline{X};\overline{Y})$*.*

(ii) *If* $\varphi(\overline{X};\overline{Y})$ *admits a (finite-state) synchronous realizer, then* $\mathsf{SMSO} \vdash \exists\overline{Y}\,\neg\neg\varphi(\overline{X};\overline{Y})$*.*

The correctness part (i) of Thm. 3.4 will be proved in §5 using a notion of realizability for SMSO based on automata and synchronous finite-state functions. The completeness part (ii) will be proved in §4.1, relying the completeness of the axiomatization of MSO (Thm. 2.10) together with the correctness of the negative translation $\neg\neg(-)$ (Thm. 3.3).

4 On the Representation of Mealy Machines in MSO

This section gathers several (possibly known) results related to the representation of Mealy machines in MSO. We begin in §4.1 with the completeness part of Thm. 3.4, which follows usual representations of automata in MSO (see e.g. [16, §5.3]). We then recall from [14, 12] the *Recursion Theorem*, which is a convenient tool to reason on runs of deterministic automata in MSO (§4.2). In §4.3 we state a Lemma for the correctness part of Thm. 3.4, which relies on the usual translation of MSO-formulae over *finite words* to DFA's (see e.g. [16, §3.1]). Finally, in §4.4 we give a possible strengthening of the synchronous comprehension rule of SMSO (but which is based on Büchi's Theorem 2.6).

We work with the following notion of representation.

▶ **Definition 4.1.** Let φ be a formula with free variables among $z, x_1, \dots, x_p, X_1, \dots, X_q$. We say that φ *z-represents* $F : \mathbf{2}^p \times \mathbf{2}^q \longrightarrow_{\mathbf{M}} \mathbf{2}$ if for all $n \in \mathbb{N}$, all $\overline{A} \in (\mathbf{2}^\omega)^q$, and all $\overline{k} \in (\mathbf{2}^\omega)^p$ such that $k_i \leq n$ for all $i \leq p$, we have

$$F(\overline{k},\overline{A})(n) = 1 \qquad \text{iff} \qquad \mathfrak{N} \models \varphi[n/z, \overline{k}/\overline{x}, \overline{A}/\overline{X}] \tag{3}$$

4.1 Internalizing Mealy Machines in MSO

The completeness part (ii) of Thm. 3.4 relies on the following simple fact.

▶ **Proposition 4.2.** *For every finite-state synchronous* $F : \mathbf{2}^p \longrightarrow_{\mathbf{M}} \mathbf{2}$*, one can build a deterministic uniformly bounded formula* $\delta[\overline{X}, x]$ *which* x*-represents* F*.*

Proof. The proof is a simple adaptation of the usual pattern (see e.g. [16, §5.3]). Let $F : \mathbf{2}^p \to_{\mathbf{M}} \mathbf{2}$ be induced by a Mealy machine $\mathcal{M}$. W.l.o.g. we can assume the state set of $\mathcal{M}$ to be of the form $\mathbf{2}^q$. Then F is represented by a formula of the form

$$\delta[\overline{X},x] \;:=\; \forall\overline{Q},Y\left(\left[\begin{array}{l}\forall t \leq \dot{x}\,(\mathsf{Z}(t) \to \mathsf{I}[\overline{Q}(t)]) \quad \wedge \\ \forall t,t' \leq \dot{x}\,(\mathsf{S}(t,t') \to \mathsf{H}[\overline{Q}(t),\overline{X}(t),Y(t),\overline{Q}(t')])\end{array}\right] \longrightarrow\; Y(x)\right) \tag{4}$$

where $\overline{X} = X_1, \dots, X_p$ codes sequences of inputs, Y codes sequences of outputs, and where $\overline{Q} = Q_1, \dots, Q_q$ codes runs. ◀

▶ Remark. In the proof of Prop. 4.2, since $\mathcal{M}$ is deterministic, we can assume the formula $\mathsf{I}[\overline{Q}(t)]$ to be of the form $\bigwedge_{1\leq i\leq q}[Q_i(t) \leftrightarrow \mathsf{B}_i]$ with $\mathsf{B}_i \in \{\top, \bot\}$, and, for some propositional formulae $\mathsf{O}[-,-], \overline{\mathsf{D}}[-,-]$, the formula $\mathsf{H}[\overline{Q}(t), \overline{X}(t), \overline{Y}(t), \overline{Q}(t')]$ to be of the form

$$\big(Y(t) \longleftrightarrow \mathsf{O}[\overline{Q}(t), \overline{X}(t)]\big) \quad \wedge \quad \bigwedge_{1\leq i\leq q} \big(Q_i(t') \longleftrightarrow \mathsf{D}_i[\overline{Q}(t), \overline{X}(t)]\big) \tag{5}$$

▶ **Example 4.3.** The function induced by the Mealy machine of Ex. 2.3.(c) (depicted on Fig. 1, right), is represented by a formula of the form (4), where $\overline{Q} = Q$ (since the machine has state set $\mathbf{2}$), $\overline{X} = X$, where $\mathsf{I}[-] := [(-) \leftrightarrow \bot]$ (since state 0 is initial) and

$$\mathsf{O}[Q(t), X(t)] \quad = \quad \mathsf{D}[Q(t), X(t)] \quad = \quad (\neg Q(t) \ \vee \ [Q(t) \wedge X(t)]) \tag{6}$$

The completeness of our approach to Church's synthesis is obtained as follows.

Proof of Thm. 3.4.(ii). Assume that $\varphi(\overline{X}; \overline{Y})$ admits a realizer $C : \mathbf{2}^q \longrightarrow_{\mathbf{M}} \mathbf{2}^p$. Using the Cartesian structure of $\mathbf{M}$ (Prop. 2.5), we assume $C = \overline{C} = C_1, \dots, C_p$ with $C_i : \mathbf{2}^q \to_{\mathbf{M}} \mathbf{2}$. We thus have $\mathfrak{N} \models \varphi[\overline{B}/\overline{X}, \overline{C}(\overline{B})/\overline{Y}]$ for all $\overline{B} \in (\mathbf{2}^\omega)^q \simeq (\mathbf{2}^q)^\omega$. Now, by Prop. 4.2 there are uniformly bounded (deterministic) formulae $\overline{\delta} = \delta_1, \dots, \delta_p$, with free variables among $\overline{X}, x$, and such that (3) holds for all $i = 1, \dots, p$. It thus follows that $\mathfrak{N} \models \forall \overline{X} \varphi[\overline{\delta[x]}/\overline{Y}]$. Then, by completeness (Thm. 2.10) we know that $\vdash \varphi[\overline{\delta}[x]/\overline{Y}]$ is provable in MSO, and by negative translation (Thm. 3.3) we get $\mathsf{SMSO} \vdash \neg\neg\varphi[\overline{\delta}[x]/\overline{Y}]$. We can then apply ($p$ times) the synchronous comprehension scheme of SMSO and obtain $\mathsf{SMSO} \vdash \exists \overline{Y} \neg\neg\varphi(\overline{X}; \overline{Y})$. ◀

▶ **Example 4.4.** Recall the specification (1) from [17], represented in MSO by the formula $\phi(X; Y)$ of Ex. 2.7. Write $\phi(X; Y) = \phi_0[X, Y] \wedge \phi_1[X, Y] \wedge \phi_2[X, Y]$ where

$$\begin{array}{lcl} \phi_0[X, Y] & := & \forall t\, (Xt \ \to \ Yt) \\ \phi_1[X, Y] & := & \forall t, t'\, (\mathsf{S}(t, t') \wedge \neg Yt \ \to \ Yt') \\ \phi_2[X, Y] & := & (\exists^\infty t\ \neg Xt) \ \to \ (\exists^\infty t\ \neg Yt) \end{array}$$

Note that ϕ_0 and ϕ_1 are monotonic in Y, while ϕ_2 is anti-monotonic in Y. The formula ϕ_0 is trivially realized by the identity function $\mathbf{2} \to_{\mathbf{M}} \mathbf{2}$ (see Ex. 2.3.(a)), which is itself represented by the deterministic uniformly bounded formula $\delta_0[X, x] := (x \mathrel{\dot\in} X)$. For ϕ_1 (which asks Y not to have two consecutive occurrences of 0), consider

$$\delta_1[X, x] \quad := \quad \delta_0[X, x] \ \vee \ \exists t \mathrel{\dot\leq} x[\mathsf{S}(t, x) \wedge \neg X(t)]$$

We have $\mathsf{MSO} \vdash \phi_0[X, \delta_1[x]/Y]$ since $\delta_0 \vdash_{\mathsf{MSO}} \delta_1$ and moreover $\mathsf{MSO} \vdash \phi_1[X, \delta_1[x]/Y]$ since

$$\mathsf{S}(t, t')\,,\ \neg Xt\,,\ \neg\exists u(\mathsf{S}(u, t) \wedge \neg Xu) \ \vdash_{\mathsf{MSO}} \ Xt' \ \vee \ \exists t''(\mathsf{S}(t'', t') \wedge \neg Xt'')$$ ◀

The case of ϕ_2 in Ex. 4.4 is more complex. The point is that $\phi_2[\delta_1[x]/Y]$ does not hold because if $\forall^\infty t\ \neg Xt$ (that is if X remains constantly 0 from some time on), then δ_1 will output no 1's. On the other hand, the machine of Ex. 2.3.(c) involves an internal state, and can be represented using a fixpoint formula of the form (4). Reasoning on such formulae is easier with more advanced tools on MSO, that we provide in §4.2.

4.2 The Recursion Theorem

Theorem 3.4.(ii) ensures that SMSO is able to handle all solvable instances of Church's synthesis, but it gives no hint on how to actually produce proofs. When reasoning on fixpoint formulae as those representing Mealy machines in Prop. 4.2, a crucial role is played by the *Recursion Theorem* for MSO [14] (see also [12]). The Recursion Theorem says that MSO allows to define predicates by well-founded induction w.r.t. the relation $\dot{<}$ defined as $(x \dot{<} y) := (x \dot{\leq} y \wedge \neg(x \doteq y))$. Given a formula ψ and variables X and x, we say that ψ *is x-recursive in X* when the following formula $\mathsf{Rec}^x_X(\psi)$ holds:

$$\mathsf{Rec}^x_X(\psi) \quad := \quad \forall z \forall Z, Z' \left(\forall y \dot{<} z \, [Zy \longleftrightarrow Z'y] \;\longrightarrow\; [\psi[Z/X, z/x] \longleftrightarrow \psi[Z'/X, z/x]]\right)$$

(where z, Z, Z' do not occur free in ψ). For $\psi[X, x]$ x-recursive in X, the Recursion Theorem says that, provably in MSO, the equation $\forall x(Xx \longleftrightarrow \psi[X, x])$ has a unique solution.

▶ **Theorem 4.5** (Recursion Theorem [14]). *If* $\mathsf{MSO} \vdash \mathsf{Rec}^x_X(\psi)$ *then*

$$\begin{array}{lll} & \forall z \left(Zz \longleftrightarrow \forall X \left[\forall x \dot{\leq} z(Xx \leftrightarrow \psi) \longrightarrow Xz\right]\right) & \vdash_{\mathsf{MSO}} \quad \forall x \, (Zx \longleftrightarrow \psi[Z/X]) \\ \textit{and} & \forall x(Zx \longleftrightarrow \psi[Z/X]) \, , \, \forall x(Z'x \longleftrightarrow \psi[Z'/X]) & \vdash_{\mathsf{MSO}} \quad \forall x \, (Zx \longleftrightarrow Z'x) \end{array}$$

▶ **Example 4.6.**

(a) W.r.t. the representation used in Prop. 4.2, let $\theta[\overline{X}, \overline{Q}, Y, x]$ be

$$\forall t \dot{\leq} x(\mathsf{Z}(t) \longrightarrow \mathsf{I}[\overline{Q}(t)]) \quad \wedge \quad \forall t, t' \dot{\leq} x \, (\mathsf{S}(t, t') \longrightarrow \mathsf{H}[\overline{Q}(t), \overline{X}(t), Y(t), \overline{Q}(t')])$$

so that $\delta[\overline{X}, x] = \forall \overline{Q}, Y \left(\theta[\overline{X}, \overline{Q}, Y, x] \to Y(x)\right)$. The Recursion Theorem 4.5 implies that, provably in MSO, for all $\overline{X}$ there are unique predicates $\overline{Q}, Y$ s.t. $\forall x.\theta[\overline{X}, \overline{Q}, Y, x]$. Indeed, assuming I and H are as in (5) we have that $\theta[\overline{X}, \overline{Q}, Y, x]$ is equivalent to $\theta^o[\overline{Q}, \overline{X}, Y, x] \wedge \bigwedge_{1 \leq i \leq q} \theta_i[\overline{Q}, \overline{X}, Y, x]$, where

$$\begin{array}{rrcl} & \theta^o[\overline{X}, \overline{Q}, Y, x] & := & \forall t \dot{\leq} x \, (Y_i(t) \;\longleftrightarrow\; \mathsf{O}_i[\overline{Q}(t), \overline{X}(t)]) \\ & \theta_i[\overline{X}, \overline{Q}, Y, x] & := & \forall t \dot{\leq} x \, (Q_i(t) \;\longleftrightarrow\; \tilde{\theta}_i[\overline{Q}, \overline{X}, t]) \\ \text{with} & \tilde{\theta}_i[\overline{X}, \overline{Q}, t] & := & (\mathsf{Z}(t) \wedge \mathsf{B}_i) \;\vee\; \exists u \dot{\leq} t \left(\mathsf{S}(u, t) \wedge \mathsf{D}_i[\overline{Q}(u), \overline{X}(u)]\right) \end{array}$$

Now, apply Thm. 4.5 to $\mathsf{O}[\overline{Q}(t), \overline{X}(t)]$ (resp. $\tilde{\theta}_i$), which is t-recursive in Y (resp. in Q_i).

(b) The machine of Ex. 2.3.(c) is represented as in (a) with O and D given by (6) (see Ex. 4.3, recalling that the machine as only two states). Hence MSO proves that for all X there are unique Q, Y such that $\forall x.\theta[X, Q, Y, x]$. Continuing now Ex. 4.4, let

$$\delta_2[X, x] \quad := \quad \forall Q, Y \, (\theta[X, Q, Y, x] \longrightarrow Y(x))$$

It is not difficult to derive $\mathsf{MSO} \vdash \phi_0[\delta_2[x]/Y] \wedge \phi_1[\delta_2[x]/Y]$. In order to show $\phi_2[\delta_2[y]/Y]$, one has to prove $\exists^\infty t \; (\neg Xt) \;\vdash_{\mathsf{MSO}}\; \exists^\infty t \; \exists Q, Y(\theta[X, Q, Y, t] \;\wedge\; \neg Yt)$. Thanks to Thm. 4.5, this follows from $\forall x.\theta[X, Q, Y, x] \, , \, \exists^\infty t \; (\neg Xt) \;\vdash_{\mathsf{MSO}}\; \exists^\infty t \; (\neg Yt)$ which itself can be derived using induction.

4.3 From Bounded Formulae to Mealy Machines

We now turn to a useful fact for part (i) of Thm. 3.4, namely, for synchronous comprehension, the extraction of finite-state synchronous functions from bounded formulae. This relies on the standard translation of MSO-formulae *over finite words* to DFA's (see e.g. [16, §3.1]).

▶ **Lemma 4.7.** *Let $\hat{\varphi}$ be a formula with free variables among $z, x_1, \dots, x_p, X_1, \dots, X_q$, and which is bounded by z. Then $\hat{\varphi}$ z-represents a finite-state synchronous $C : \mathbf{2}^p \times \mathbf{2}^q \to_{\mathbf{M}} \mathbf{2}$ induced by a Mealy machine computable from $\hat{\varphi}$.*

▶ Remark. Given $C : \mathbf{2}^p \times \mathbf{2}^q \to_{\mathbf{M}} \mathbf{2}$ z-represented by $\psi\restriction[- \dot{\leq} z]$ (with z not free in ψ), for all $n \in \mathbb{N}$, all $\overline{A} \in (\mathbf{2}^\omega)^q$ and all $\overline{k} \in (\mathbf{2}^\omega)^p$ with $k_i \leq n$, we have $C(\overline{k}, \overline{A})(n) = 1$ if and only if $\langle \overline{k}, \overline{A}\restriction(n+1)\rangle \models \psi$ (in the sense of MSO over finite words). It follows that if C is induced by a Mealy machine $\mathcal{M} = (Q, q^\imath, \partial)$, then with the DFA $\mathcal{A} := (Q \times \mathbf{2} + \{q^\imath\}, q^\imath, \partial_{\mathcal{A}}, Q \times \{1\})$ where $\partial_{\mathcal{A}}(q^\imath, \mathtt{a}) := \partial(q^\imath, \mathtt{a})$ and $\partial_{\mathcal{A}}((q, b), \mathtt{a}) := \partial(q, \mathtt{a})$, we have $C(\overline{k}, \overline{A})(n) = 1$ iff $\mathcal{A}$ accepts the finite word $\langle \overline{k}, \overline{A}\restriction(n+1)\rangle$. Hence $\mathcal{M}$ must pay the price of the non-elementary lower-bound for translating MSO-formulae over finite words to DFAs (see e.g. [7, Chap. 13]). ◀

▶ **Example 4.8.** Recall the continuous but not synchronous function P of Ex. 2.3.(d). The function P can be used to realize a predecessor function, and thus is represented (in the sense of (3)) by a formula $\varphi[X, Y, x]$ such that $\mathfrak{N} \models \varphi[A, B, n]$ iff $A = \{k+1\}$ and $B = \{k\}$ for some $k \leq n$. But φ is not equivalent to a bounded formula, since by Lem. 4.7 bounded formulae represent synchronous functions.

4.4 Internally Bounded Formulae

The synchronous comprehension scheme of MSO is motivated by Lem. 4.7, which tells that uniformly bounded formulae induce Mealy machines. However, being uniformly bounded may seem to be a strict syntactic requirement, and one may wish to relax synchronous comprehension to formulae which behave as bounded formulae, that is to formulae $\psi[\overline{X}, x]$ such that the following formula $\mathsf{B}^x_{\overline{X}}(\psi[\overline{X}, x])$ holds (where $z, \overline{Z}, \overline{Z'}$ do not occur free in ψ):

$$\mathsf{B}^x_{\overline{X}}(\psi[\overline{X}, x]) \quad := \quad \forall z \forall \overline{Z}\,\overline{Z'} (\forall y \dot{\leq} z [\overline{Zz \longleftrightarrow Z'z}] \longrightarrow [\psi[\overline{Z}/\overline{X}, z/x] \longleftrightarrow \psi[\overline{Z'}/\overline{X}, z/x]])$$

▶ **Theorem 4.9.** *If $\mathsf{MSO} \vdash \mathsf{B}^x_{\overline{X}}(\psi[\overline{X}, x])$ and the free variables of ψ are among $x, \overline{X}$, then there is a uniformly bounded formula $\hat{\varphi}[\overline{X}, x]$ which is effectively computable from ψ and such that $\mathsf{MSO} \vdash \forall \overline{X}\, \forall x\, (\psi[\overline{X}, x] \longleftrightarrow \hat{\varphi}[\overline{X}, x])$.*

▶ Remark. Theorem 4.9 relies on the decidability of MSO. Note that Thm. 4.9 in part. applies if $\mathsf{SMSO} \vdash \mathsf{B}^x_{\overline{X}}(\psi[\overline{X}, x])$. Moreover, if $\psi[X, x]$ is recursive (in the sense of §4.2), then $\mathsf{B}^x_X(\psi[X, x])$ holds, but not conversely.

5 The Realizability Interpretation of MSO

This Section presents our realizability model for SMSO, and uses it to prove Thm. 3.4.(i). Our approach to Church's synthesis *via* realizability uses automata in two different ways. First, from a *proof* $\mathscr{D}$ in SMSO of an existential formula $\exists \overline{Y} \varphi(\overline{X}; \overline{Y})$, one can compute a finite-state synchronous realizer $\overline{F}$ of $\varphi(\overline{X}; \overline{Y})$. Second, the adequacy of realizability (and in particular the correctness of $\overline{F}$ w.r.t. $\varphi(\overline{X}; \overline{Y})$) is *proved* using automata for $\varphi(\overline{X}; \overline{Y})$ obtained by McNaughton's Theorem, but these automata do not have to be built concretely.

5.1 Uniform Automata

The adequacy of realizability will be proved using the notion of *uniform automata* (adapted from [13]). In our context, uniform automata are essentially usual non-deterministic automata, but in which non-determinism is expressed *via* an explicitly given set of *moves*. This allows a simple inheritance of the Cartesian structure of synchronous functions (Prop. 2.5), and

thus to interpret the positive existentials of SMSO similarly as usual (weak) sums of type theory. In particular, the set of moves $M(\mathcal{A})$ of an automaton $\mathcal{A}$ interpreting a formula φ will exhibit the strictly positive existentials of φ as $M(\mathcal{A}) = M(\varphi)$ where

$$M(\alpha) \simeq M(\neg\varphi) \simeq \mathbf{1} \qquad M(\varphi \wedge \psi) \simeq M(\varphi) \times M(\psi) \qquad M(\exists(-)\,\varphi) \simeq \mathbf{2} \times M(\varphi) \tag{7}$$

▶ **Definition 5.1** ((Non-Deterministic) Uniform Automata)**.** A (non-deterministic) *uniform automaton* $\mathcal{A}$ over Σ (notation $\mathcal{A} : \Sigma$) has the form

$$\mathcal{A} \quad = \quad (Q_{\mathcal{A}}\,,\, q^{\imath}_{\mathcal{A}}\,,\, M(\mathcal{A})\,,\, \partial_{\mathcal{A}}\,,\, \Omega_{\mathcal{A}}) \tag{8}$$

where $Q_{\mathcal{A}}$ is the finite set of *states*, $q^{\imath}_{\mathcal{A}} \in Q_{\mathcal{A}}$ is the *initial state*, $M(\mathcal{A})$ is the finite non-empty set of *moves*, the *acceptance condition* $\Omega_{\mathcal{A}}$ is an ω-regular subset of $Q^{\omega}_{\mathcal{A}}$, and the *transition function* $\partial_{\mathcal{A}}$ has the form

$$\partial_{\mathcal{A}} \quad : \quad Q_{\mathcal{A}} \times \Sigma \quad \longrightarrow \quad M(\mathcal{A}) \quad \longrightarrow \quad Q_{\mathcal{A}}\,.$$

A *run* of $\mathcal{A}$ on an ω-word $B \in \Sigma^{\omega}$ is an ω-word $R \in M(\mathcal{A})^{\omega}$. We say that R is *accepting* (notation $R \Vdash \mathcal{A}(B)$) if $(q_k)_{k\in\mathbb{N}} \in \Omega_{\mathcal{A}}$ for the sequence of states $(q_k)_{k\in\mathbb{N}}$ defined as $q_0 := q^{\imath}_{\mathcal{A}}$ and $q_{k+1} := \partial_{\mathcal{A}}(q_k, B(k), R(k))$. We say that $\mathcal{A}$ *accepts* B if there exists an accepting run of $\mathcal{A}$ on B, and we let $\mathcal{L}(\mathcal{A})$ be the set of ω-words accepted by $\mathcal{A}$.

Following the usual terminology, an automaton $\mathcal{A}$ as in (8) is *deterministic* if $M(\mathcal{A}) \simeq \mathbf{1}$.

Let us now sketch how uniform automata will be used in our realizability interpretation of SMSO. First, by adapting to our context usual constructions on automata (§5.2), to each MSO-formula φ with free variables among (say) $\overline{X} = X_1, \ldots, X_q$, we associate a uniform automaton $[\![\varphi]\!]$ over $\mathbf{2}^q$ (Fig. 5). Then, from an SMSO-derivation $\mathscr{D}$ of a sequent (say) $\varphi \vdash \psi$ (with free variables among $\overline{X}$ as above), we will extract a finite-state synchronous function $F_{\mathscr{D}} : \mathbf{2}^q \times M([\![\varphi]\!]) \longrightarrow_{\mathbf{M}} M([\![\psi]\!])$, such that $F_{\mathscr{D}}(\overline{B}, R) \Vdash [\![\psi]\!](\overline{B})$ whenever $R \Vdash [\![\varphi]\!](\overline{B})$. In the case of $\vdash \exists Y\, \phi(\overline{X}; Y)$, the finite-state realizer $F_{\mathscr{D}}$ will be of the form $\langle C, G\rangle$ with C and G finite-state synchronous functions $C : \mathbf{2}^q \longrightarrow_{\mathbf{M}} \mathbf{2}$ and $G : \mathbf{2}^q \longrightarrow_{\mathbf{M}} M(\phi)$ such that $G(\overline{B}) \Vdash [\![\phi]\!](\overline{B}, C(\overline{B}))$ for all $\overline{B}$. This motivates the following notion.

▶ **Definition 5.2** (The Category Aut_{Σ})**.** For each alphabet Σ, the category Aut_{Σ}, has automata $\mathcal{A} : \Sigma$ as objects. Morphisms F from $\mathcal{A}$ to $\mathcal{B}$ (notation $\mathcal{A} \Vdash F : \mathcal{B}$) are finite-state synchronous maps $F : \Sigma \times M(\mathcal{A}) \longrightarrow_{\mathbf{M}} M(\mathcal{B})$ such that $F(B, R) \Vdash \mathcal{B}(B)$ whenever $R \Vdash \mathcal{A}(B)$.

▶ Remark.

(a) Note that if $\mathcal{B} \Vdash F : \mathcal{A}$ for some F, then $\mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{A})$.

(b) One could also consider the category AUT_{Σ} defined as Aut_{Σ}, but with maps not required to be finite-state. All statements of this Section hold for AUT_{Σ}, but for Cor. 5.10, which would lead to non necessarily finite-state realizers and would not give Thm. 3.4.(i).

(c) Uniform automata are a variation of usual automata on ω-words, which is convenient for our purposes, namely the adequacy of our realizability interpretation. Hence, while it would have been possible to define uniform automata with any of the usual acceptance condition (see e.g. [16]), we lose nothing by assuming their acceptance condition to be given by arbitrary ω-regular sets.

5.2 Constructions on Automata

We gather here constructions on uniform automata that we will need to interpret MSO formulae. First, automata are closed under the following operation of *finite substitution*.

▶ **Proposition 5.3.** *Given $\mathcal{A} : \Sigma$ and a function $\mathtt{f} : \Gamma \to \Sigma$, let $\mathcal{A}[\mathtt{f}] : \Gamma$ be the automaton identical to $\mathcal{A}$, but with $\partial_{\mathcal{A}[\mathtt{f}]}(q, \mathtt{b}, u) := \partial_{\mathcal{A}}(q, \mathtt{f}(\mathtt{b}), u)$. Then $B \in \mathcal{L}(\mathcal{A}[\mathtt{f}])$ iff $\mathtt{f} \circ B \in \mathcal{L}(\mathcal{A})$.*

▶ **Example 5.4.** Assume $\mathcal{A}$ interprets a formula φ with free variables among $\overline{X}$, so that $\overline{B} \in \mathcal{L}(\mathcal{A})$ iff $\mathfrak{N} \models \varphi[\overline{B}/\overline{X}]$. Then φ is also a formula with free variables among $\overline{X}, \overline{Y}$, and we have $\overline{B}\overline{B'} \in \mathcal{L}(\mathcal{A}[\pi])$ iff $\mathfrak{N} \models \varphi[\overline{B}/\overline{X}/\overline{B'}/\overline{Y}]$, where $\pi : \overline{X} \times \overline{Y} \to \overline{X}$ is a projection.

The Cartesian structure of $\mathbf{M}$ lifts to Aut_Σ. This gives the interpretation of conjunctions.

▶ **Proposition 5.5.** *For each Σ, the category Aut_Σ has finite products. Its terminal object is the automaton $\mathbf{I} = (\mathbf{1}, \bullet, \mathbf{1}, \partial_{\mathbf{I}}, \mathbf{1}^\omega)$, where $\partial_{\mathbf{I}}(-,-,-) = \bullet$. Binary products are given by*

$$\begin{array}{rrcl} & \mathcal{A} \times \mathcal{B} & := & (Q_{\mathcal{A}} \times Q_{\mathcal{B}}\,,\, (q^\imath_{\mathcal{A}}, q^\imath_{\mathcal{B}})\,,\, M(\mathcal{A}) \times M(\mathcal{B})\,,\, \partial\,,\, \Omega) \\ \textit{where} & \partial((q_{\mathcal{A}}, q_{\mathcal{B}}), \mathtt{a}, (u, v)) & := & (\partial_{\mathcal{A}}(q_{\mathcal{A}}, \mathtt{a}, u)\ ,\ \partial_{\mathcal{B}}(q_{\mathcal{B}}, \mathtt{a}, v)) \end{array}$$

and where $(q_n, q'_n)_n \in \Omega$ iff $((q_n)_n \in \Omega_{\mathcal{A}}$ and $(q'_n)_n \in \Omega_{\mathcal{B}})$. Note that Ω is ω-regular since $\Omega_{\mathcal{A}}$ and $\Omega_{\mathcal{B}}$ are ω-regular (see e.g. [10, Ex. I.11.3.7]). Moreover, $\mathcal{L}(\mathbf{I}) = \Sigma^\omega$ and $\mathcal{L}(\mathcal{A} \times \mathcal{B}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$.

Uniform automata are equipped with the obvious adaptation of the usual projection on non-deterministic automata, which interprets existentials. Given a uniform automaton $\mathcal{A} : \Sigma \times \Gamma$, its *projection on* Σ is the automaton

$$(\exists_\Gamma \mathcal{A} : \Sigma) \ := \ (Q_{\mathcal{A}}\,,\, q^\imath_{\mathcal{A}}\,,\, \Gamma \times M(\mathcal{A})\,,\, \partial\,,\, \Omega_{\mathcal{A}}) \quad \text{where} \quad \partial(q, \mathtt{a}, (\mathtt{b}, u)) \ := \ \partial_{\mathcal{A}}(q, (\mathtt{a}, \mathtt{b}), u)$$

▶ **Proposition 5.6.** *Given $\mathcal{A} : \Sigma \times \Gamma$ and $\mathcal{B} : \Sigma$, the realizers $\mathcal{B} \Vdash F : \exists_\Gamma \mathcal{A}$ are exactly the $\mathbf{M}$-pairs $\langle C, G\rangle$ of synchronous functions $C : \Sigma \times M(\mathcal{B}) \to_{\mathbf{M}} \Gamma$ and $G : \Sigma \times M(\mathcal{B}) \to_{\mathbf{M}} M(\mathcal{A})$ such that $G(B, R) \Vdash \mathcal{A}\langle B, C(B, R)\rangle$ for all $B \in A^\omega$ and all $R \Vdash \mathcal{B}(B)$.*

The negation $\neg(-)$ of SMSO is interpreted by an operation $\sim(-)$ on uniform automata which involves McNaughton's Theorem 2.8.

▶ **Proposition 5.7.** *Given a uniform automaton $\mathcal{A} : \Sigma$, there is a uniform* deterministic *$\sim\mathcal{A} : \Sigma$ such that $B \in \mathcal{L}(\sim\mathcal{A})$ iff $B \notin \mathcal{L}(\mathcal{A})$.*

5.3 The Realizability Interpretation

Consider a formula φ with free variables among $\overline{x} = x_1, \ldots, x_p$ and $\overline{X} = X_1, \ldots, X_q$. Its interpretation $[\![\varphi]\!]_{\overline{x},\overline{X}}$ is the uniform automaton over $\mathbf{2}^p \times \mathbf{2}^q$ defined by induction over φ in Fig. 5, where $\mathcal{A}_\alpha$ is a deterministic uniform automaton for the atomic formula α, $\mathsf{Sing} : \mathbf{2}$ is a deterministic uniform automaton accepting the $B \in \mathbf{2}^\omega \simeq \mathcal{P}(\mathbb{N})$ such that B is a singleton, and π, π' are suitable projections. We write $[\![\varphi]\!]$ when $\overline{x}, \overline{X}$ are irrelevant or understood from the context. Note that the set of moves $M(\varphi)$ of $[\![\varphi]\!]$ indeed satisfies (7), so in particular $[\![\delta]\!]$ is deterministic for a deterministic δ. Moreover, as expected we get:

▶ **Proposition 5.8.** *Given an MSO-formula φ with free variables among $\overline{x} = x_1, \ldots, x_p$ and $\overline{X} = X_1, \ldots, X_q$, for all $\overline{k} \in (\mathbf{2}^\omega)^p \simeq (\mathbf{2}^p)^\omega$ and all $\overline{B} \in (\mathbf{2}^\omega)^q \simeq (\mathbf{2}^q)^\omega$, we have $(\overline{k}, \overline{B}) \in \mathcal{L}([\![\varphi]\!]_{\overline{x},\overline{X}})$ iff $\mathfrak{N} \models \varphi[\overline{k}/\overline{x}, \overline{B}/\overline{X}]$.*

Let $\varphi_1, \ldots, \varphi_n, \varphi$ be MSO-formulae with free variables among $\overline{x} = x_1, \ldots, x_p$ and $\overline{X} = X_1, \ldots, X_q$. Then we say that a synchronous function

$$F \quad : \quad \mathbf{2}^p \times \mathbf{2}^q \times M(\varphi_1) \times \cdots \times M(\varphi_n) \quad \longrightarrow_{\mathbf{M}} \quad M(\varphi)$$

realizes the sequent $\varphi_1, \ldots, \varphi_n \vdash \varphi$ (notation $\varphi_1, \ldots, \varphi_n \Vdash F : \varphi$ or $\overline{\varphi} \Vdash F : \varphi$) if

$$[\![\varphi_1]\!]_{\overline{x},\overline{X}} \times \cdots \times [\![\varphi_n]\!]_{\overline{x},\overline{X}} \quad \Vdash \quad F \quad : \quad [\![\varphi]\!]_{\overline{x},\overline{X}}$$

$$[\![\alpha]\!]_{\overline{x},\overline{X}} := \mathcal{A}_\alpha[\pi] \qquad [\![\neg\psi]\!]_{\overline{x},\overline{X}} := \sim[\![\psi]\!]_{\overline{x},\overline{X}} \qquad [\![\exists X\,\psi]\!]_{\overline{x},\overline{X}} := \exists_{\mathbf{2}}([\![\psi]\!]_{\overline{x},\overline{X},X})$$
$$[\![\psi_1 \wedge \psi_2]\!]_{\overline{x},\overline{X}} := [\![\psi_1]\!]_{\overline{x},\overline{X}} \times [\![\psi_2]\!]_{\overline{x},\overline{X}} \qquad [\![\exists x\,\psi]\!]_{\overline{x},\overline{X}} := \exists_{\mathbf{2}}(\mathsf{Sing}[\pi] \times [\![\psi]\!]_{\overline{x},x,\overline{X}}[\pi'])$$

Figure 5 Interpretation of MSO-Formulae as Uniform Automata.

▶ **Theorem 5.9** (Adequacy). *Let $\overline{\varphi}, \varphi$ be* MSO*-formulae with variables among $\overline{x}, \overline{X}$. From an* SMSO*-derivation $\mathscr{D}$ of $\overline{\varphi} \vdash \varphi$, one can compute an* **M***-morphism $F_{\mathscr{D}}$ s.t. $\overline{\varphi} \Vdash_{\overline{x},\overline{X}} F_{\mathscr{D}} : \varphi$.*

Proof. The proof is by induction on derivations. Note that if $\overline{\varphi} \vdash_{\mathsf{SMSO}} \varphi$, then $\overline{\varphi} \models_{\mathfrak{N}} \varphi$. In part., for all rules whose conclusion is of the form $\overline{\varphi} \vdash \delta$ with δ deterministic, it follows from Prop. 5.8 and (7) that the unique **M**-map with codomain $M(\delta) \simeq \mathbf{1}$ (and with appropriate domain) is a realizer. A similar argument applies to the *Ex Falso* rule (elimination of $\bot$), but in this case the realizer of $\overline{\varphi} \vdash \varphi$ is not canonical, and elimination of equality is direct from Prop. 5.8. Adequacy for synchronous comprehension is deferred to §5.3.1. As for the rules of Fig. 4, the first two rules follow from the fact that **M** is a category with finite limits (Prop. 2.5), and the rules for conjunction (resp. existentials) follow from Prop. 5.5 (resp. Prop. 5.6). It remains the rules $\overline{\overline{\varphi} \vdash \exists y\,\mathsf{Z}(y)}$ and $\overline{\overline{\varphi} \vdash \exists y\,\mathsf{S}(x,y)}$ of Fig. 3. For the latter, we use the Mealy machine depicted on Fig. 1 (left) (Ex. 2.3.(b)) together with the fact that $\mathsf{S}(-,-)$ is deterministic. The case of the former is similar and simpler. ◀

Adequacy of realizability, together with Prop. 5.6, directly gives Thm. 3.4.(i).

▶ **Corollary 5.10** (Thm. 3.4.(i)). *Given a derivation $\mathscr{D}$ in* SMSO *of $\vdash \exists\overline{Y}\,\varphi(\overline{X};\overline{Y})$ with $\overline{X} = X_1,\dots,X_q$ and $\overline{Y} = Y_1,\dots,Y_p$, we have $F_{\mathscr{D}} = \langle \overline{C}, G\rangle$ where $\overline{C} = C_1,\dots,C_p$ with $C_i : \mathbf{2}^q \longrightarrow_{\mathbf{M}} \mathbf{2}$ and $\mathfrak{N} \models \varphi(\overline{B}, \overline{C}(\overline{B}))$ for all $\overline{B} \in (\mathbf{2}^\omega)^q \simeq (\mathbf{2}^q)^\omega$.*

5.3.1 Realization of Synchronous Comprehension

We now turn to the adequacy of the synchronous comprehension rule. It directly follows from the existence of finite-state characteristic functions for bounded formulae (Lem. 4.7) and from the following semantic substitution lemma, which allows, given a synchronous function $C_{\hat{\varphi}}$ y-represented by $\hat{\varphi}$, to lift a realizer of $\psi[\hat{\varphi}[y]/Y]$ into a realizer of $\exists Y\,\psi$.

▶ **Lemma 5.11.** *Let $\overline{x} = x_1,\dots,x_p$ and $\overline{X} = X_1,\dots,X_q$. Let $\hat{\varphi}$ be a formula with free variables among $y, \overline{X}$, and assume that $\hat{\varphi}$ y-represents $C_{\hat{\varphi}} : \mathbf{2}^q \longrightarrow_{\mathbf{M}} \mathbf{2}$. Then for every* MSO*-formula ψ with free variables among $\overline{x}, \overline{X}$, there is a finite-state synchronous function*

$$H_\psi \quad : \quad M(\psi[\hat{\varphi}[y]/Y]) \quad \longrightarrow_{\mathbf{M}} \quad M(\psi)$$

such that for all $\overline{k} \in (\mathbf{2}^\omega)^p$, all $\overline{A} \in (\mathbf{2}^\omega)^q$ and all $R \in M(\psi)^\omega$, we have

$$R \Vdash [\![\psi[\hat{\varphi}[y]/Y]]\!]_{\overline{x},\overline{X}}(\overline{k},\overline{A}) \qquad \Longrightarrow \qquad H_\psi(R) \Vdash [\![\psi]\!]_{\overline{x},\overline{X},Y}(\overline{k},\overline{A}, C_{\hat{\varphi}}(\overline{A})) \tag{9}$$

Adequacy of synchronous comprehension then directly follows.

▶ **Lemma 5.12.** *Let ψ with free variables among $\overline{x}, \overline{X}, Y$ and let $\hat{\varphi}$ be a formula with free variables among $y, \overline{X}$ and which is uniformly bounded by y. Then there is a finite-state realizer $\psi[\hat{\varphi}[y]/Y] \Vdash_{\overline{x},\overline{X}} F : \exists Y\,\psi$, effectively computable from ψ and φ.*

Proof. Let $C_{\hat{\varphi}}$ satisfying (3) be given by Lem. 4.7, and let H_ψ satisfying (9) be given by Lem. 5.11. It then directly follows from Prop. 5.6 and Len. 5.11 that $\psi[\hat{\varphi}[y]/Y] \Vdash_{\overline{x},\overline{X}} \langle C_{\hat{\varphi}} \circ [\pi], H_\psi \circ [\pi']\rangle : \exists Y\,\psi$, where π, π' are suitable projections. ◀

6 Conclusion

In this paper, we revisited Church's synthesis *via* an automata-based realizability interpretation of an intuitionistic proof system SMSO for MSO on ω-words, and we demonstrated that our approach is sound and complete, in the sense of Thm. 3.4. As it stands, this approach must still pay the price of the non-elementary lower-bound for the translation of MSO formulae over finite words to DFA's (see the Remark after Lem. 4.7, §4.3) and the system SMSO is limited by its set of connectives and its restricted induction scheme.

Further Works. First, following the approach of [13], SMSO could be extended with primitive universal quantifications and implications as soon as one goes to a *linear* deduction system. In particular, primitive universals and implications would allow to extend the logic with atomic formulae for Mealy machines with defining axioms of the form (4). We expect this to give better lower bounds w.r.t. completeness (for each solvable instance of Church's synthesis, to provide proofs with realizers of a reasonable complexity). Among other outcomes of going to a linear deduction system, following [13] we expect similar proof-theoretical properties as with the usual *Dialectica* interpretation (see e.g. [8]), such as realizers of linear Markov rules and choices schemes. On the other hand, we do not know yet if effective computations of modulus of uniform continuity could be pertinent for Church's synthesis (e.g. for a non-linear Markov rule). Moreover, we expect that a linear variant of MSO on finite words could help (for some classes of formulae) to mitigate the Remark of §4.3.

The case of induction is more complex. One possibility to have finite-state realizers for a more general induction rule would be to rely on saturation techniques for regular languages. Another possibility, which may be of practical interest, is to follow the usual Curry-Howard approach and allow possibly infinite-state realizers.

Another direction of future work is to incorporate specific reasoning principles on Mealy machines. For instance, a translation from (a subsystem of) [2] could be interesting.

Acknowledgements. We thank the anonymous referees for helpful comments.

References

1 R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive (1) designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012.

2 M. Bonsangue, J. Rutten, and A. Silva. Coalgebraic logic and synthesis of Mealy machines. In *Proceedings of FOSSACS'08*, pages 231–245. Springer, 2008.

3 J. R. Büchi. On a Decision Methond in Restricted Second-Order Arithmetic. In E. Nagel et al., editor, *Logic, Methodology and Philosophy of Science (Proc. 1960 Intern. Congr.)*, pages 1–11. Stanford Univ. Press, 1962.

4 J. R. Büchi and L. H. Landweber. Solving Sequential Conditions by Finite-State Strategies. *Transation of the American Mathematical Society*, 138:367–378, 1969.

5 A. Church. Applications of recursive arithmetic to the problem of circuit synthesis. In *Summaries of the SISL*, volume 1, pages 3–50. Cornell Univ., 1957.

6 J.-Y. Girard. *Interprétation Fonctionnelle et Élimination des Coupures de l'Arithmétique d'Ordre Supérieur*. PhD thesis, Université Paris 7, 1972.

7 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.

8 U. Kohlenbach. *Applied Proof Theory: Proof Interpretations and their Use in Mathematics*. Springer Monographs in Mathematics. Springer, 2008.

9 R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966.

10 D. Perrin and J.-É. Pin. *Infinite Words: Automata, Semigroups, Logic and Games*. Pure and Applied Mathematics. Elsevier, 2004.

11 M. O. Rabin. Automata on infinite objects and Church's Problem. *Amer. Math. Soc.*, 1972.

12 C. Riba. A model theoretic proof of completeness of an axiomatization of monadic second-order logic on infinite words. In *Proceedings of IFIP-TCS'12*, 2012.

13 C. Riba. A Dialectica-Like Approach to Tree Automata. Available on HAL (hal-01261183), `https://hal.archives-ouvertes.fr/hal-01261183`, 2016.

14 D. Siefkes. *Decidable Theories I: Büchi's Monadic Second Order Successor Arithmetic*, volume 120 of *LNM*. Springer, 1970.

15 The Coq Developement Team. *The Coq Proof Assistant Reference Manual*, 2016. `http://coq.inria.fr/`.

16 W. Thomas. Languages, Automata, and Logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume III, pages 389–455. Springer, 1997.

17 W. Thomas. Solution of Church's Problem: A tutorial. *New Perspectives on Games and Interaction*, 5:23, 2008.

Combinatorial Flows and Their Normalisation

Lutz Straßburger

Inria Saclay, Palaiseau, France

Abstract

This paper introduces combinatorial flows that generalize combinatorial proofs such that they also include cut and substitution as methods of proof compression. We show a normalization procedure for combinatorial flows, and how syntactic proofs are translated into combinatorial flows and vice versa.

1998 ACM Subject Classification F.4.1 [Mathematical Logic] Proof Theory

Keywords and phrases proof equivalence, cut elimination, substitution, deep inference

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.31

1 Introduction

Proof theory is a central area of theoretical computer science, as it can provide the foundations not only for logic programming and functional programming, but also for the formal verification of software. Yet, despite the crucial role played by formal proofs, we have no proper notion of proof identity telling us when two proofs are "the same". This is very different from other areas of mathematics, like group theory, where two groups are "the same" if they are isomorphic, or topology, where two spaces are "the same" if they are homeomorphic.

The problem is that proofs are usually presented by syntactic means, and depending on the chosen syntactic formalism, "the same" proof can look very different. In fact, one can say that at the current state of art, *proof theory is not a theory of proofs but a theory of proof formalisms*. This means that the first step must be to find ways to describe proofs independent of the formalisms, i.e., we need "canonical representations" which do not rely on some particular syntax of a chosen deductive formalism. For this reason, we also speak of "syntax-free" presentation of proofs.

The earliest attempts for such "syntax-free" proof presentations were Andrews' *matings* [1] and Bibel's *matrix proofs* [3] for propositional logic. However, checking correctness of a mating or matrix proof is exponential, and thus not more efficient than starting a proof search from scratch. Furthermore, matings and matrix proofs are not able to address proof normalization procedures like cut elimination.

Girard's *proof-nets* for linear logic [11] were the first syntax-free proof presentation able to address these two issues. Proof nets can be seen as graphs that abstract away from the syntax of the sequent calculus, such that it is decidable in polynomial time whether a given such graph is indeed a correct proof, and such that the normalization of proofs via cut elimination is simpler in proof-nets than in the sequent calculus.

Clearly, it became a research question whether such a notion of proof-net is also possible for classical logic. An immediate idea is to use exactly the same notion of proof-net as for linear logic [22, 28]. However, these proof-nets depend on a specific form of Gentzen's sequent calculus. They are neither able to capture proofs written in other sequent calculi, like G3c [32], nor other formalisms, like analytic tableaux or resolution.

This problem was addressed by *B-nets* [21], which exhibit a confluent cut elimination procedure and can capture proofs in most standard proof formalisms. However, their correctness criterion is exponential and the cut elimination cannot be lifted to the sequent calculus.

This issue has been addressed by *atomic flows* [12, 13] that are more fine-grained than Boolean nets and that have a number of different cut elimination procedures that can all be lifted to a deep inference proof system. However, atomic flows do not have a correctness criterion. In fact, the work by Das [9] shows that there cannot be a polynomial correctness criterion for atomic flows, if integer factoring is hard for *P/poly*. The *C-nets* of [29], which are similar to atomic flows, but additionally form a closed category, have same problem.

Only the *combinatorial proofs* by Hughes [16] have a polynomial correctness criterion and are independent of any syntactic formalism. Cuts in combinatorial proofs are represented by formulas of the shape $A \wedge \bar{A}$ in the conclusion [17]. The cut elimination in [17] is then based on a form of projection combined with atomic substitution, and this construction largely inspired our vertical composition in Section 6.

Anyway, none of the existing "syntax-free" proof presentations can deal with proofs using *extension* or *substitution* [6, 20], which are, like the *cut*, methods of proof compression. They are mainly studied in the area of proof complexity, and only recently have received attention from structural proof theory [5, 30, 25].

The main contribution of this paper is a notion of "syntax-free" proof presentation that (1) comes with a polynomial correctness criterion, (2) is independent of the syntax of proof formalisms (like sequent calculi, tableaux systems, resolution, Frege systems, or deep inference systems), and (3) can handle cut and substitution, and their elimination. The main idea is to combine the advantages of combinatorial proofs and of atomic flows, and add a notion of substitution. Point (1) above is stated in Theorem 22, Point (3) is carried out in Sections 5 and 6. For Point (2), we sketch in Section 8 how deep inference proofs are translated into combinatorial flows. The technical report [31] also sketches how sequent proofs and Frege proofs can be translated into combinatorial flows.[1] In a future work we will show how this can be done for other fomalisms, like analytic tableaux or resolution. The proposed notion of proof identity here is that *"two proofs are the same if they have the same combinatorial flow"*.

2 Preliminaries on combinatorial proofs

Combinatorial proofs have been introduced by Hughes in [16] as a way to present proofs of classical logic independent of a syntactic proof system. To make our paper self-contained, we recall here the basic definitions.

We consider formulas (denoted by capital Latin letters $A, B, C, \ldots$) in negation normal form (NNF), generated from a countable set $\mathcal{V} = \{a, b, c, \ldots\}$ of (propositional) variables by the following grammar:

$$A, B ::= a \mid \bar{a} \mid A \wedge B \mid A \vee B \tag{1}$$

where $\bar{a}$ is the negation of a. The negation can then be defined for all formulas via $\bar{\bar{a}} = a$ and the De Morgan laws $\overline{A \vee B} = \bar{A} \wedge \bar{B}$ and $\overline{A \wedge B} = \bar{A} \vee \bar{B}$. Then it follows that $\bar{\bar{A}} = A$ for all formulas A. An *atom* is a variable or its negation. We use $\mathcal{A}$ to denote the set of all

[1] The report [31] also contains more technical details and missing proofs.

atoms. Sometimes we use $A \Rightarrow B$ as abbreviation for $\bar{A} \vee B$, and $A \Leftrightarrow B$ as abbreviation for $(A \Rightarrow B) \wedge (B \Rightarrow A)$.

A *sequent* Γ is a multiset of formulas, written as a list separated by comma:

$$\Gamma = A_1, A_2, \ldots, A_n \tag{2}$$

We write $\bar{\Gamma}$ to denote the sequent $\bar{A}_1, \bar{A}_2, \ldots, \bar{A}_n$. We define the *size* of a sequent Γ, denoted by $|\Gamma|$, to be the number of atom occurrences in it. We write $\wedge\Gamma$ (resp. $\vee\Gamma$) for the conjunction (resp. disjunction) of the formulas in Γ.

▶ Remark. For simplicity we do not include the constants $\top$ and $\bot$ (for *truth* and *falsum*, respectively) into the language. We can always recover them by letting $\top = a_0 \vee \bar{a}_0$ and $\bot = a_0 \wedge \bar{a}_0$ for some fresh variable a_0. Note that in this respect, classical logic is different from linear logic, where the removal of the constants does indeed change the logic.

Before we can discus the notion of combinatorial proof, we need some preliminary definitions.

▶ **Definition 1.** A *(simple) graph* $\mathfrak{G} = \langle V_\mathfrak{G}, E_\mathfrak{G} \rangle$ consists of a set of *vertices* $V_\mathfrak{G}$ and a set of *edges* $E_\mathfrak{G}$ which are two-element subsets of $V_\mathfrak{G}$. If $E_\mathfrak{G}$ is not a set but a multiset, we call $\mathfrak{G}$ a *multigraph*. We omit the index $\mathfrak{G}$ when it is clear from context. For $v, w \in V$ we write vw for $\{v, w\}$. The *size* of a graph $\mathfrak{G}$, denoted by $|\mathfrak{G}|$ is $|V_\mathfrak{G}| + |E_\mathfrak{G}|$. A *graph homomorphism* $f \colon \mathfrak{G} \to \mathfrak{G}'$ is a function from $V_\mathfrak{G}$ to $V_{\mathfrak{G}'}$ such that $vw \in E_\mathfrak{G}$ implies $f(v)f(w) \in E_{\mathfrak{G}'}$. A simple graph $\mathfrak{G}$ is called a *cograph* if it does not contain four distinct vertices u, v, w, z with $uv, vw, wz \in E$ and $vz, zu, uw \notin E$. For a set L, a graph $\mathfrak{G}$ is *L-labeled* if every vertex of $\mathfrak{G}$ is associated with an element in L, called its *label*. For two disjoint graphs $\mathfrak{G} = \langle V, E \rangle$ and $\mathfrak{G}' = \langle V', E' \rangle$, we define the operations *union* $\mathfrak{G} \vee \mathfrak{G}' = \langle V \cup V', E \cup E' \rangle$ and *join* $\mathfrak{G} \wedge \mathfrak{G}' = \langle V \cup V', E \cup E' \cup \{vv' \mid v \in V, v' \in V'\} \rangle$. If $\mathfrak{G}$ and $\mathfrak{G}'$ are L-labeled graphs, then so are $\mathfrak{G} \vee \mathfrak{G}'$ and $\mathfrak{G} \wedge \mathfrak{G}'$ where every vertex keeps its original label. For a simple graph $\mathfrak{G} = \langle V, E \rangle$, also define its *negation* $\bar{\mathfrak{G}} = \langle V, \{vw \mid v \neq w, vw \notin E\} \rangle$. If $\mathfrak{G}$ is an $\mathcal{A}$-labeled graph (where $\mathcal{A}$ is the set of atoms) then all labels are negated in $\bar{\mathfrak{G}}$. For two homomorphisms $f_1 \colon \mathfrak{G}_1 \to \mathfrak{G}'_1$ and $f_2 \colon \mathfrak{G}_2 \to \mathfrak{G}'_2$ such that $V_{\mathfrak{G}_1} \cap V_{\mathfrak{G}_2} = \varnothing$, we define $f_1 \vee f_2 \colon \mathfrak{G}_1 \vee \mathfrak{G}_2 \to \mathfrak{G}'_1 \vee \mathfrak{G}'_2$ to be the *union* of the two homomorphisms f_1 and f_2, and $f_1 \wedge f_2 \colon \mathfrak{G}_1 \wedge \mathfrak{G}_2 \to \mathfrak{G}'_1 \wedge \mathfrak{G}'_2$ to be their *join*.

▶ **Construction 2.** *If we associate to each atom a a single vertex labeled with a then every formula A uniquely determines a graph $\mathfrak{G}(A)$ that is constructed via the operations $\wedge$ and $\vee$. For a sequent $\Gamma = A_1, A_2, \ldots, A_n$, we define $\mathfrak{G}(\Gamma) = \mathfrak{G}(\vee\Gamma) = \mathfrak{G}(A_1) \vee \mathfrak{G}(A_2) \vee \cdots \vee \mathfrak{G}(A_n)$.*

Note that this construction entails that $\overline{\mathfrak{G}(A)} = \mathfrak{G}(\bar{A})$.

▶ **Lemma 3.** *For two formulas A and B, we have $\mathfrak{G}(A) = \mathfrak{G}(B)$ iff A and B are equivalent modulo associativity and commutativity of $\wedge$ and $\vee$:*

$$\begin{array}{ll} A \wedge (B \wedge C) = (A \wedge B) \wedge C & A \wedge B = B \wedge A \\ A \vee (B \vee C) = (A \vee B) \vee C & A \vee B = B \vee A \end{array} \tag{3}$$

Proof. Immediately from Construction 2. ◀

▶ **Example 4.** Let $A = (a \wedge (b \vee \bar{c})) \vee (c \wedge \bar{d})$ then $\bar{A} = (\bar{a} \vee (\bar{b} \wedge c)) \wedge (\bar{c} \vee d)$. Below are the two graphs $\mathfrak{G}(A)$ and $\mathfrak{G}(\bar{A}) = \overline{\mathfrak{G}(A)}$:

(4)

The following is well-known. It can already be found in [10] (see also [24, 26]).

▶ **Proposition 5.** *A graph $\mathfrak{G}$ is a cograph iff it can be constructed from a formula via Construction 2.*

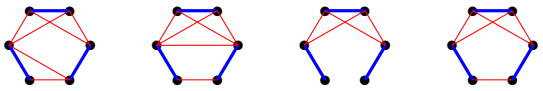
An important consequence of this and Lemma 3 is that for each cograph $\mathfrak{G}$ there is a unique (up to associativity and commutativity) formula tree determining $\mathfrak{G}$. We denote this formula tree by $F(\mathfrak{G})$.

▶ **Definition 6.** Let $\mathfrak{G} = \langle V, E\rangle$ be a cograph, let $V' \subseteq V$, and let E' be the restriction of E to V'. We say that $\mathfrak{G}' = \langle V', E'\rangle$ is a *subcograph* of $\mathfrak{G}$ iff for all $v \in V'$ and $w_1, w_2 \in V\backslash V'$ we have $vw_1 \in E$ iff $vw_2 \in E$. In this case we also say that V' *induces a subcograph.*

It follows immediately from the definition that any subcograph is indeed a cograph. Furthermore, $\mathfrak{G}'$ is a subcograph of $\mathfrak{G}$ iff $F(\mathfrak{G}')$ is a subformula of $F(\mathfrak{G})$.

▶ **Definition 7.** Let $\mathfrak{G} = \langle V_\mathfrak{G}, E_\mathfrak{G}\rangle$ be a multigraph. A set $B_\mathfrak{G} \subseteq E_\mathfrak{G}$ of edges is called a *matching* if no two edges in $B_\mathfrak{G}$ are adjacent. A matching $B_\mathfrak{G}$ is *perfect* if every vertex $v \in V_\mathfrak{G}$ is incident to an edge in $B_\mathfrak{G}$. An *R&B-graph* $\mathfrak{G} = \langle V_\mathfrak{G}, R_\mathfrak{G}, B_\mathfrak{G}\rangle$ is a triple such that $\langle V_\mathfrak{G}, R_\mathfrak{G} \uplus B_\mathfrak{G}\rangle$ is a multigraph such that $B_\mathfrak{G}$ is a perfect matching and $\langle V_\mathfrak{G}, R_\mathfrak{G}\rangle$ is a simple graph (i.e., $R_\mathfrak{G}$ is not allowed to have multiple edges). We will use the notation $\mathfrak{G}^\downarrow$ for the simple graph $\langle V_\mathfrak{G}, R_\mathfrak{G}\rangle$. An *R&B-cograph* is an R&B-graph $\mathfrak{G} = \langle V_\mathfrak{G}, R_\mathfrak{G}, B_\mathfrak{G}\rangle$ where $\mathfrak{G}^\downarrow = \langle V_\mathfrak{G}, R_\mathfrak{G}\rangle$ is a cograph.

As before, we omit the index $\mathfrak{G}$ when it is clear from context. Following [27] we will draw B-edges in blue/bold, and R-edges in red/regular. Below are four examples:



(5)

Also the next two definitions are taken from [27].

▶ **Definition 8.** A path (resp. cycle) in a multigraph is said to be *elementary* if it does not contain two equal vertices (resp. but the first and last one). A path $\mathcal{P}$ in a graph with a matching B is *alternating* if the edges of $\mathcal{P}$ are alternately in B and not in B. Let $\mathfrak{G} = \langle V, R, B\rangle$ be an R&B-graph. An *æ-path* in $\mathfrak{G}$ is an elementary alternating path in $\langle V, R \uplus B\rangle$. An *æ-cycle* in $\mathfrak{G}$ is an elementary alternating cycle of even length in $\langle V, R \uplus B\rangle$, so that when turning around the cycle, the edges are still alternately in B and not in B. A *chord* of a path (resp. cycle) is an edge that is not part of the path (resp. cycle) but connects two vertices of the path (resp. cycle). An æ-path (resp. æ-cycle) is called *chordless* iff it does not have any chords.

Note that chords for æ-paths, resp. æ-cycles, are always R-edges because B is a perfect matching. We are now ready to present a central concept for R&B-cographs:

▶ **Definition 9.** An R&B-cograph $\mathfrak{G} = \langle V, R, B\rangle$ is *critically chorded* if $\langle V, R \uplus B\rangle$ does not contain any chordless æ-cycle, and any two vertices in V are connected by a chordless æ-path.

In the examples in (5), the first one is not an R&B-cograph, the other three are. The second one has a chordless æ-cycle, and the third one has no chordless æ-path between the lowermost vertices. Only the last one is a critically chorded R&B-cograph.

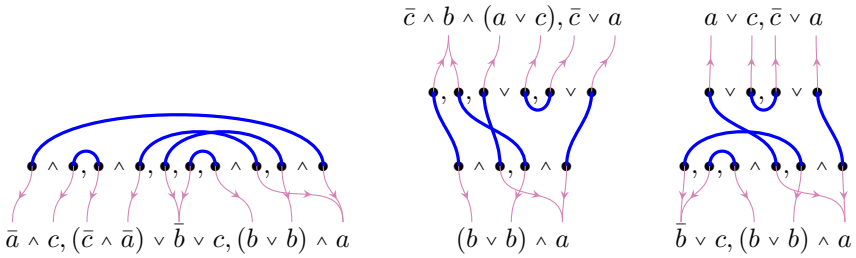


Figure 1 Examples of simple combinatorial flows (the cographs are obtained via Construction 2).

▶ **Definition 10.** Let $\mathfrak{C} = \langle V, R, B \rangle$ be an R&B-graph and $f \colon \mathfrak{C}^{\downarrow} \to \mathfrak{G}$ be a graph-homomorphism and let $\mathfrak{G}$ be $\mathcal{A}$-labeled (where $\mathcal{A}$ is the set of atoms). We say f is *axiom-preserving* iff $wv \in B$ implies that the labels of $f(w)$ and $f(v)$ are dual to each other.

▶ **Definition 11.** A graph homomorphism f is a *skew fibration*, denoted as $f \colon \mathfrak{G} \rightarrowtail \mathfrak{G}'$, if for every $v \in V_{\mathfrak{G}}$ and $w' \in V_{\mathfrak{G}'}$ with $f(v)w' \in E'_{\mathfrak{G}}$ there is a $w \in V_{\mathfrak{G}}$ with $vw \in E_{\mathfrak{G}}$ and $f(w)w' \notin E'_{\mathfrak{G}}$.

We are now ready to give the definition of a combinatorial proof together with the main result of [16].

▶ **Definition 12.** A *combinatorial proof* of a sequent Γ consists of a non-empty critically chorded R&B-cograph $\mathfrak{C}$ and an axiom-preserving skew-fibration $f \colon \mathfrak{C}^{\downarrow} \rightarrowtail \mathfrak{G}(\Gamma)$.

▶ Remark. Our presentation of the condition on the cograph in a combinatorial proof deters from Hughes' [16] and follows Retoré's [27] instead. The reason is that Retoré makes the relation to proof nets of linear logic [8] explicit. Also note, that the condition on the cograph $\mathfrak{C}^{\downarrow}$ given by Hughes [16, 17] is weaker than ours. It is equivalent to our condition of $\mathfrak{C}$ not containing any chordless æ-cycle. In terms of linear logic, this is equivalent to the correctness condition for MLL proof nets with the mix-rule [27]. In our presentation here we also add the connectedness via chordless æ-paths in order to reject mix.

The two main results of [16] are that combinatorial proofs are sound and complete with respect to classical logic, and that they form a proof system in the sense of Cook and Reckhow [6].

▶ **Theorem 13** ([16]). *A formula is a theorem of classical propositional logic iff it has a combinatorial proof.*

▶ **Theorem 14** ([16]). *Given a formula A, some R&B-graph $\mathfrak{C}$, and some mapping $f \colon \mathfrak{C}^{\downarrow} \to \mathfrak{G}(A)$, we can decide in polynomial time in the size of the input that $\mathfrak{C}$ and f form a combinatorial proof of A.*

3 Combinatorial flows

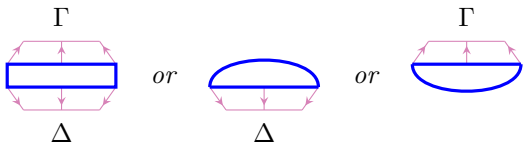
▶ **Definition 15.** Given two sequents Γ and Δ, a *simple (combinatorial) flow* ϕ from Γ to Δ, denoted by $\phi \colon \Gamma \vdash \Delta$, is a combinatorial proof for the sequent $\bar{\Gamma}, \Delta$. We write $\phi \colon \circ \vdash \Delta$ (resp. $\phi \colon \Gamma \vdash \circ$) if Γ (resp. Δ) is empty.[2] Let ϕ be given by the R&B-cograph $\mathfrak{C}$ and skew fibration $f \colon \mathfrak{C}^{\downarrow} \rightarrowtail \mathfrak{G}(\bar{\Gamma}, \Delta)$. Then the *size* of ϕ, denoted by $|\phi|$, is defined to be $|\mathfrak{C}^{\downarrow}| + |\Gamma| + |\Delta|$.

[2] Note that it cannot happen that both Γ and Δ are empty.

▶ **Lemma 16.** *Let $\mathfrak{C}$, $\mathfrak{G}_1$, and $\mathfrak{G}_2$ be cographs and let $f\colon \mathfrak{C} \rightarrowtail \mathfrak{G}_1 \vee \mathfrak{G}_2$ be a skew fibration. Then there are cographs $\mathfrak{C}_1$ and $\mathfrak{C}_2$ and graph homomorphisms $f_1\colon \mathfrak{C}_1 \rightarrowtail \mathfrak{G}_1$ and $f_2\colon \mathfrak{C}_2 \rightarrowtail \mathfrak{G}_2$ such that $\mathfrak{C} = \mathfrak{C}_1 \vee \mathfrak{C}_2$ and $f = f_1 \vee f_2$.*

▶ **Notation 17.** *This lemma allows us to depict simple combinatorial flows in the following way. Let $\phi\colon \Gamma \vdash \Delta$ be given, let $f\colon \mathfrak{C}^{\downarrow} \rightarrowtail \overline{\mathfrak{G}(\Gamma)} \vee \mathfrak{G}(\Delta)$ be the defining skew fibration, and let $\mathfrak{C}_\Gamma$ and $\mathfrak{C}_\Delta$ be the cographs determined by Lemma 16 (i.e., $\mathfrak{C}^{\downarrow} = \mathfrak{C}_\Gamma \vee \mathfrak{C}_\Delta$). If we write $F(\overline{\mathfrak{C}_\Gamma})$ and $F(\mathfrak{C}_\Delta)$ for the formula trees corresponding to the cographs $\overline{\mathfrak{C}_\Gamma}$ and $\mathfrak{C}_\Delta$, respectively, then we can write ϕ by writing Γ, $F(\overline{\mathfrak{C}_\Gamma})$, $F(\mathfrak{C}_\Delta)$, and Δ above each other, draw the B-edges and indicate the mapping f by thin (thistle) arrows. Figure 1 shows some examples. For better readability, we allow in $F(\overline{\mathfrak{C}_\Gamma})$ outermost $\wedge$ to be replaced by comma, and in $F(\mathfrak{C}_\Delta)$ outermost $\vee$ to be replaced by comma. Note that the three flows in Figure 1 are just "flipped variants" of each other, i.e., are defined by the same R&B-cograph and skew fibration.*

Schematically we can depict simple combinatorial flows as follows:

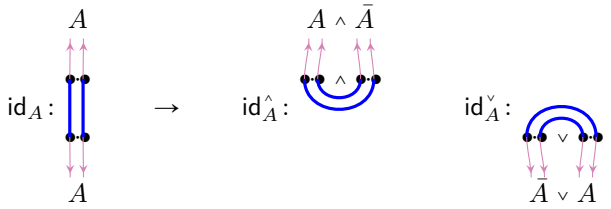


where the middle and the right picture are used to indicate that Γ or Δ, respectively, are empty.

▶ **Lemma 18.** *Let Γ, Δ, Σ be sequents. There is a one-to-one correspondence between the simple combinatorial flows $\Gamma \vdash \Sigma, \Delta$ and $\bar{\Sigma}, \Gamma \vdash \Delta$. In particular, for any three formulas A, B, C, there is a one-to-one correspondence between the simple combinatorial flows $A \vdash B \vee C$ and $\bar{B} \wedge A \vdash C$.*

Proof. This follows immediately from Definition 15. ◀

▶ **Observation 19.** *For every formula A, we have a simple combinatorial flow $\mathsf{id}_A\colon A \vdash A$, that we call the* identity flow *and that is defined by the identity skew fibration $\overline{\mathfrak{G}(A)} \vee \mathfrak{G}(A) \rightarrowtail \mathfrak{G}(\bar{A}, A)$ where the matching is defined such that it pairs each vertex in $V_{\mathfrak{G}(A)}$ to itself in the copy $V_{\overline{\mathfrak{G}(A)}}$. When applying Lemma 18 to id_A we get two simple combinatorial flows $\mathsf{id}_A^{\wedge}\colon A \wedge \bar{A} \vdash \circ$ and $\mathsf{id}_A^{\vee}\colon \circ \vdash \bar{A} \vee A$, as depicted below:*



(6)

▶ **Definition 20.** A *substitution* is a mapping σ from propositional variables to formulas such that $\sigma(a) \neq a$ for only finitely many a.

We write $A\sigma$ for the formula obtained from applying the substitution σ to the formula A. If $\sigma = \{a_1 \mapsto B_1, \ldots, a_n \mapsto B_n\}$ we also write $A[a_1/B_1, \ldots, a_n/B_n]$ for $A\sigma$. This normally means that not only is each occurrence of a_i in A is replaced by B_i in A, but also each occurrence of $\bar{a}_i$ is replaced by $\bar{B}_i$. Then, for substitution proof into proofs, we also need a notation for formula substitutions in which a variable a and its dual $\bar{a}$ are not replaced by

dual formulas. In this case we write $A[a_1/B_1, \bar{a}_1/C_1, \ldots, a_n/B_n, \bar{a}_n/C_n]$ for the formula that is obtained from A by simultaneously replacing every a_i by B_i and every $\bar{a}_i$ by C_i for each $i \in \{1, \ldots, n\}$.

▶ **Definition 21.** The set of *combinatorial flows* is defined inductively as follows:

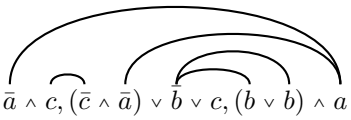
- A simple combinatorial flow $\phi\colon A \vdash B$ is a combinatorial flow.
- If $\phi\colon A \vdash B$ and $\psi\colon C \vdash D$ are combinatorial flows then so are $\phi \wedge \psi\colon A \wedge B \vdash C \wedge D$ and $\phi \vee \psi\colon A \vee C \vdash B \vee D$. This operation is called *horizontal composition*.
- If $\phi\colon \Gamma \vdash A$ and $\psi\colon A \vdash \Delta$ are combinatorial flows then $\phi \blacklozenge \psi\colon \Gamma \vdash \Delta$ is a combinatorial flow. This operation is called *vertical composition*, *concatenation*, or *cut*.
- If $\phi\colon \Gamma \vdash \Delta$ and $\psi\colon C \vdash D$ are combinatorial flows then $\phi[a/\psi]\colon \Gamma[a/C, \bar{a}/\bar{D}] \vdash \Delta[a/D, \bar{a}/\bar{C}]$ is a combinatorial flow. This operation is called *substitution*.

The *size* of a combinatorial flow ϕ, denoted by $|\phi|$, is defined to be the sum of the sizes of all simple combinatorial flows occurring in ϕ.

▶ **Theorem 22.** *Combinatorial flows form a proof system (in the sense of [6]). In particular, checking correctness of a combinatorial flow can be done is polynomial time.*

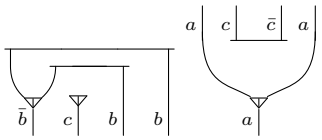
Proof. This follows immediately from Theorem 14, Definition 15, and Definition 21. ◀

▶ Remark. Theorem 22 provides the main advantage of combinatorial flows over B-nets and N-nets [21] and atomic flows [12, 13]. For a simple combinatorial flow $\phi\colon \circ \vdash \Gamma$, we can immediately obtain the corresponding N-net by forgetting the cograph $\langle V, R\rangle$ and connecting the atoms of Γ according to the (undirected) paths given by f and B. The example below is obtained from the first flow in Figure 1:



(7)

The corresponding B-net is obtained by forgetting the multiplicity of the edges. In the example in (7), the B-net is identical to the N-net. For translating a simple combinatorial flow $\phi\colon \Delta \vdash \Gamma$ into an atomic flow, we not only forget the cograph $\langle V, R\rangle$ but also the structure of Γ and the order of the atoms in Γ. We only look at the paths given by f and B and keep track of which atoms are in Γ and which ones are in Δ. Here is the third example in Figure 1 translated into an atomic flow:



A substitution-free combinatorial flow can straightforwardly translated into atomic flows since they can be composed horizontally and vertically. However, in each translation, critical information is lost, such that it becomes impossible to recover the proof from an N-net or an atomic flow in polynomial time.

▶ **Definition 23.** A combinatorial flow is *normal* if it is a simple combinatorial flow. It is *cut-free* if the composition operation $\blacklozenge$ is not used in it, and it is *substitution-free* if the substitution operation is not used in it.

Normalization of a combinatorial flow means therefore to remove the operations defined in Definition 21. The following four sections are dedicated to this.

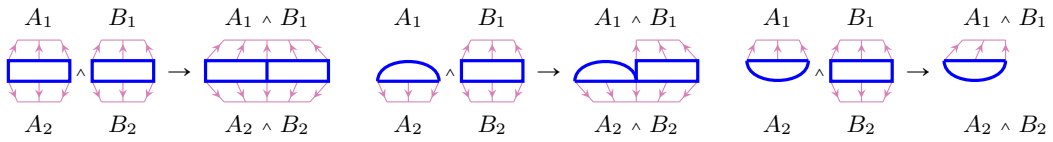


Figure 2 Conjunction of simple combinatorial flows.

4 Normalization I: Binary Connectives

▶ **Lemma 24.** *Let $\phi\colon A_1 \vdash A_2$ and $\psi\colon B_1 \vdash B_2$ be simple combinatorial flows. Then there are simple combinatorial flows $\chi\colon A_1 \wedge B_1 \vdash A_2 \wedge B_2$ and $\xi\colon A_1 \vee B_1 \vdash A_2 \vee B_2$, such that $|\chi| \leqslant |\phi| + |\psi|$ and $|\xi| \leqslant |\phi| + |\psi|$.*

Proof. Let $\mathfrak{C}$ and $\mathfrak{D}$ be the R&B-cographs for ϕ and ψ, respectively, and let $f\colon \mathfrak{C}^{\downarrow} \rightarrowtail \mathfrak{G}(\bar{A}_1) \vee \mathfrak{G}(A_2)$ and $g\colon \mathfrak{D}^{\downarrow} \rightarrowtail \mathfrak{G}(\bar{B}_1) \vee \mathfrak{G}(B_2)$ be their defining skew fibrations. Then, let $\mathfrak{C}_1$ and $\mathfrak{C}_2$ be the subgraphs of $\mathfrak{C}^{\downarrow}$, and $f_1\colon \mathfrak{C}_1 \to \mathfrak{G}(\bar{A}_1)$ and $f_2\colon \mathfrak{C}_2 \to \mathfrak{G}(A_2)$ be the corresponding restrictions of f, obtained via Lemma 16. Similarly, let $\mathfrak{D}_1$ and $\mathfrak{D}_2$ be the corresponding subgraphs of $\mathfrak{D}^{\downarrow}$, and g_1 and g_2 the corresponding restrictions of g.

The simple flow $\chi\colon A_1 \wedge B_1 \vdash A_2 \wedge B_2$ can now be given by the R&B-cograph $\mathfrak{H}$ and skew fibration $h\colon \mathfrak{H}^{\downarrow} \rightarrowtail \mathfrak{G}(\overline{A_1 \wedge B_1}, A_2 \wedge B_2)$ which are defined as follows:

- If $\mathfrak{C}_2$ and $\mathfrak{D}_2$ are both not empty, then we define $\mathfrak{H}^{\downarrow} = \mathfrak{D}_1 \vee \mathfrak{C}_1 \vee (\mathfrak{C}_2 \wedge \mathfrak{D}_2)$, and $B_{\mathfrak{H}} = B_{\mathfrak{C}} \cup B_{\mathfrak{D}}$, and $h = g_1 \vee f_1 \vee (f_2 \wedge g_2)$. To see that this is well-defined, note that $\mathfrak{G}(\overline{A_1 \wedge B_1}, A_2 \wedge B_2)$ is the same as $\mathfrak{G}(\bar{B}_1) \vee \mathfrak{G}(\bar{A}_1) \vee (\mathfrak{G}(A_2) \wedge \mathfrak{G}(B_2))$.
- If $\mathfrak{C}_2$ is empty then $\mathfrak{C}_1 = \mathfrak{C}^{\downarrow}$ and we define $\mathfrak{H} = \mathfrak{C}$ and let $h = f$.
- If $\mathfrak{D}_2$ is empty and $\mathfrak{C}_2$ is not, then then $\mathfrak{D}_1 = \mathfrak{D}^{\downarrow}$ and we define $\mathfrak{H} = \mathfrak{D}$ and let $h = g$.

Then, $\mathfrak{H}$ is an R&B-cograph (by construction) and it is critically chorded. In the first case the situation is the same as in the $\otimes$-rule for MLL-proof nets (see [27]) and in the other two cases it is trivial. It also trivially follows that h is axiom preserving. Therefore it only remains to show that h is indeed a skew fibration. For this, observe that $g_1 \vee f_1 \vee (f_2 \wedge g_2)$ fails to be a skew fibration only if one of $\mathfrak{C}_2$ or $\mathfrak{D}_2$ is empty. On the other hand, f is a skew-fibration from $\mathfrak{C}^{\downarrow}$ to $\mathfrak{G}(\bar{B}_1) \vee \mathfrak{G}(\bar{A}_1) \vee (\mathfrak{G}(A_2) \wedge \mathfrak{G}(B_2))$ if no vertex of $\mathfrak{C}$ is mapped to $\mathfrak{G}(A_2)$, i.e., $\mathfrak{C}_2$ is empty. Dually, we can define the simple flow $\xi\colon A_1 \vee B_1 \vdash A_2 \vee B_2$. ◀

▶ Remark. Note that it is crucial to check whether $\mathfrak{C}_2$ or $\mathfrak{D}_2$ are empty, whereas for $\mathfrak{C}_1$ and $\mathfrak{D}_1$, this is irrelevant. The difference is shown in Figure 2. Note also that there is an arbitrary choice to make when both $\mathfrak{C}_2$ and $\mathfrak{D}_2$ are empty.

5 Normalization II: Substitution

▶ **Lemma 25.** *Let $\phi\colon \Gamma \vdash \Delta$ and $\psi\colon C \vdash D$ be simple combinatorial flows. Then there is a simple combinatorial flow $\phi'\colon \Gamma[a/C, \bar{a}/\bar{D}] \vdash \Delta[a/D, \bar{a}/\bar{C}]$.*

This is depicted on the left of Figure 3. The basic idea of the construction is as follows: The simple combinatorial flow $\phi\colon \Gamma \vdash \Delta$ consists of simple paths ←—→, and each simple path in ϕ whose endpoints are occurrences of a or $\bar{a}$ are replaced according to Figure 4. To define this more formally, we first need the notion of substitution in a graph.

▶ **Construction 26.** *Let $\mathfrak{C}$ and $\mathfrak{D}$ be disjoint graphs, and let x be a vertex in $\mathfrak{C}$. With $\mathfrak{C}[x/\mathfrak{D}]$ we denote the graph whose vertex set is $V = V_{\mathfrak{C}} \backslash \{x\} \cup V_{\mathfrak{D}}$ and whose edge set is*

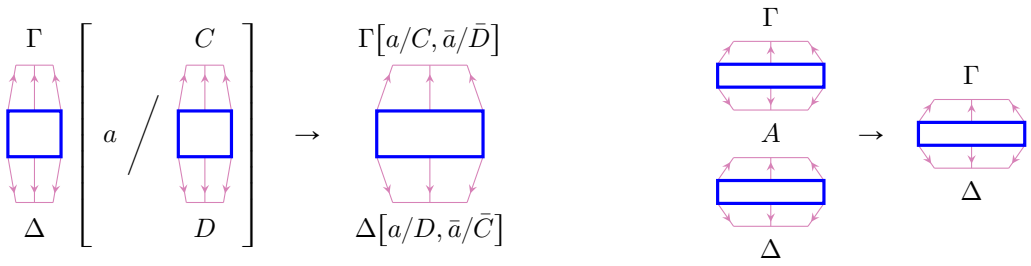


Figure 3 Left: Substitution elimination Right: cut elimination.

$E = E_{\mathfrak{C}} \backslash \{xz \mid z \in V_{\mathfrak{C}}\} \cup \{yz \mid y \in V_{\mathfrak{D}}, xz \in E_{\mathfrak{C}}\}$. *In other words, we remove* x *from* $\mathfrak{C}$ *and replace it by* $\mathfrak{D}$*, such that we have an edge from a remaining vertex* y *in* $\mathfrak{C}$ *to all vertices in* $\mathfrak{D}$*, whenever there was an edge from* y *to* x *in* $\mathfrak{C}$ *before.*

▶ **Lemma 27.** *If* $\mathfrak{C}$ *and* $\mathfrak{D}$ *are cographs and* $x \in V_{\mathfrak{C}}$*, then* $\mathfrak{C}[x/\mathfrak{D}]$ *is also a cograph.*

Proof. If we take the formula tree for $\mathfrak{C}$, remove the leaf x, and replace it by the formula tree of $\mathfrak{D}$, we obtain a formula tree for $\mathfrak{C}[x/\mathfrak{D}]$, which is therefore a cograph by Proposition 5. ◀

▶ **Construction 28.** *In Construction 26 we substituted graphs for* vertexes *in other graphs. Now we use this to substitute R&B-graphs for* B-edges *in other R&B-graphs. Let* $\mathfrak{C}$ *and* $\mathfrak{D}$ *be disjoint R&B-graphs, and let* $x, y \in V_{\mathfrak{C}}$ *with* $xy \in B_{\mathfrak{C}}$*. Furthermore, let* $\mathfrak{D}^{\downarrow} = \mathfrak{D}_1 \vee \mathfrak{D}_2$*. We now define the R&B-graph* $\mathfrak{H} = \mathfrak{C}[xy/\langle \mathfrak{D}_1 \vee \mathfrak{D}_2, B_{\mathfrak{D}} \rangle] = \langle V_{\mathfrak{H}}, R_{\mathfrak{H}}, B_{\mathfrak{H}} \rangle$ *as follows. We let* $\langle V_{\mathfrak{H}}, R_{\mathfrak{H}} \rangle = \mathfrak{C}^{\downarrow}[x/\mathfrak{D}_1][y/\mathfrak{D}_2]$*, applying Construction 26 twice, and let* $B_{\mathfrak{H}} = B_{\mathfrak{C}} \backslash \{xy\} \cup B_{\mathfrak{D}}$*. In other words,* x *is replaced by* $\mathfrak{D}_1$ *and* y *by* $\mathfrak{D}_2$*, and the* B*-edge* xy *is removed and replaced by the matching* $B_{\mathfrak{D}}$*.*

▶ **Lemma 29.** *If* $\mathfrak{C}$ *and* $\mathfrak{D}$ *are R&B-cographs with* $xy \in B_{\mathfrak{C}}$ *and* $\mathfrak{D}^{\downarrow} = \mathfrak{D}_1 \vee \mathfrak{D}_2$ *then* $\mathfrak{H} = \mathfrak{C}[xy/\langle \mathfrak{D}_1 \vee \mathfrak{D}_2, B_{\mathfrak{D}} \rangle]$ *also is an R&B-cograph. Furthermore, if* $\mathfrak{C}$ *and* $\mathfrak{D}$ *are both critically chorded, then so is* $\mathfrak{H}$*.*

Proof. The graph $\mathfrak{H}$ is a cograph for the same reason as in Lemma 27. Now assume by way of contradiction that $\mathfrak{H}$ is not critically chorded. First, assume there is a chordless æ-cycle $\mathcal{C}$. If all vertices of $\mathcal{C}$ are inside $V_{\mathfrak{C}}$ or all inside $V_{\mathfrak{D}}$, we have immediately a contradiction to $\mathfrak{C}$ and $\mathfrak{D}$ having no chordless æ-cycle. So, the cycle $\mathcal{C}$ must contain vertices from $V_{\mathfrak{C}}$ and $V_{\mathfrak{D}}$. Since by construction all B-edges are fully contained in $\mathfrak{C}$ or in $\mathfrak{D}$, we must have an R-edge participating in $\mathcal{C}$ and connecting a vertex $u \in V_{\mathfrak{C}}$ to a vertex $z \in V_{\mathfrak{D}}$. Let $v \in V_{\mathfrak{C}}$ be the unique vertex with $uv \in B_{\mathfrak{C}}$. However, since $uz \in R_{\mathfrak{H}}$, we must by construction also have $vz \in R_{\mathfrak{H}}$ which is a chord for $\mathcal{C}$. Contradiction. For showing that that any two vertices in $\mathfrak{H}$ are connected by a chordless path, we can proceed similarly. ◀

Proof of Lemma 25. Let ϕ and ψ as above and let $\Gamma' = \Gamma[a/C, \bar{a}/\bar{D}]$ and $\Delta' = \Delta[a/D, \bar{a}/\bar{C}]$. For constructing the simple flow $\phi' \colon \Gamma' \vdash \Delta'$, let $\mathfrak{C}$ and $\mathfrak{D}$ be the R&B-cographs for ϕ and ψ, respectively, and let $f \colon \mathfrak{C}^{\downarrow} \rightarrowtail \mathfrak{G}(\bar{\Gamma}, \Delta)$ and $g \colon \mathfrak{D}^{\downarrow} \rightarrowtail \mathfrak{G}(\bar{C}, D)$ be their corresponding skew fibrations. For brevity, we write $\mathfrak{G}$ for $\mathfrak{G}(\bar{\Gamma}, \Delta)$, and $\mathfrak{G}'$ for $\mathfrak{G}(\bar{\Gamma}', \Delta')$. Next, let $\mathfrak{D}_{\bar{C}}$ and $\mathfrak{D}_D$ be the two cographs obtained from $\mathfrak{D}^{\downarrow}$ via Lemma 16, and let $x_1, \ldots, x_n \in V_{\mathfrak{C}}$ be the vertexes that f maps to a vertex labeled $\bar{a}$ in $\mathfrak{G}$, and and let $y_1, \ldots, y_n \in V_{\mathfrak{C}}$ be all the vertexes that f maps to a vertex labeled a in $\mathfrak{G}$ – their number has to be identical, otherwise f could not

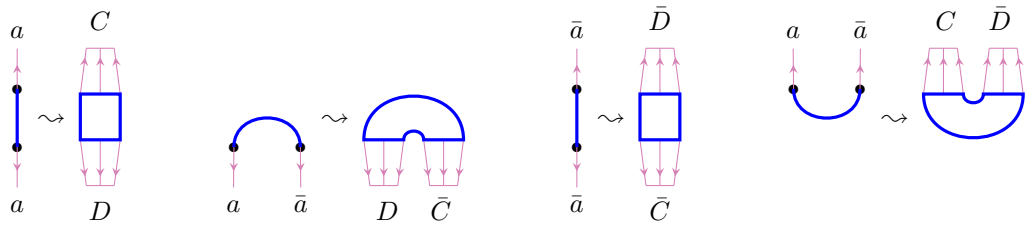


Figure 4 Substitution of simple combinatorial flows.

be axiom preserving. Without loss of generality, we can assume that $\{x_1y_1, \dots, x_ny_n\} \subseteq B_{\mathfrak{C}}$. We can now give the R&B-cograph $\mathfrak{C}'$ for ϕ' as follows:

$$\mathfrak{C}' = \mathfrak{C}[x_1y_1/\langle \mathfrak{D}_{\bar{C}} \vee \mathfrak{D}_D, B_{\mathfrak{D}} \rangle] \cdots [x_ny_n/\langle \mathfrak{D}_{\bar{C}} \vee \mathfrak{D}_D, B_{\mathfrak{D}} \rangle]$$

applying Construction 28 for each B-edge in $\mathfrak{C}$ connecting an a and an $\bar{a}$ in $\mathfrak{G}$. Finally, we define the map $f' \colon \mathfrak{C}' \rightarrowtail \mathfrak{G}'$ as follows: For every $z \in V_{\mathfrak{C}} \backslash \{x_1, \dots, x_n, y_1, \dots, y_n\}$, we have $f'(z) = f(z)$. For each x_i that is mapped by f to a $\bar{a}$, we use g to map the substituted copy of $\mathfrak{D}_{\bar{C}}$ in $\mathfrak{C}'$ to the corresponding substituted copy of $\mathfrak{G}(\bar{C})$ in $\mathfrak{G}'$. We proceed similarly for each y_i. It is easy to see that the so defined f' is indeed a skew fibration and axiom preserving. ◀

6 Normalization III: Cut

In this section, we show how cuts are eliminated, as indicated on the right of Figure 3. This is done via "projection + atomic substitution", as introduced in [17], but refined in such a way that we do not need mix and the notion of "laxness".

▶ **Lemma 30.** *Let $\phi \colon \Gamma \vdash A$ and $\psi \colon A \vdash \Delta$ be simple combinatorial flows. Then there is a simple combinatorial flow $\chi \colon \Gamma \vdash \Delta$.*

Before we give the construction of χ, we need first to establish some preliminary properties on skew fibrations and the composition of R&B-cographs.

▶ **Lemma 31.** *Let $\mathfrak{C}$, $\mathfrak{D}$, $\mathfrak{G}$, $\mathfrak{H}$ be cographs.*

1. *If $f \colon \mathfrak{C} \rightarrowtail \mathfrak{G}$ is an isomorphism, then it is also a skew fibration.*
2. *The map $\mathsf{w} \colon \mathfrak{C} \rightarrowtail \mathfrak{C} \vee \mathfrak{D}$, which behaves like the identity on $\mathfrak{C}$, is a skew fibration.*
3. *The map $\mathsf{c} \colon \mathfrak{C} \vee \mathfrak{C} \rightarrowtail \mathfrak{C}$, which maps both copies of $\mathfrak{C}$ in the domain like the identity to the $\mathfrak{C}$ in the codomain, is a skew fibration.*
4. *The map $\mathsf{m} \colon (\mathfrak{C} \wedge \mathfrak{D}) \vee (\mathfrak{G} \wedge \mathfrak{H}) \rightarrowtail (\mathfrak{C} \vee \mathfrak{G}) \wedge (\mathfrak{D} \vee \mathfrak{H})$, which maps each of $\mathfrak{C}$, $\mathfrak{D}$, $\mathfrak{G}$, and $\mathfrak{H}$ identically to itself, is a skew fibration.*
5. *If $f \colon \mathfrak{C} \rightarrowtail \mathfrak{G}$ and $g \colon \mathfrak{D} \rightarrowtail \mathfrak{H}$ are skew fibrations, then so are $f \vee g \colon \mathfrak{C} \vee \mathfrak{D} \rightarrowtail \mathfrak{G} \vee \mathfrak{H}$ and $f \wedge g \colon \mathfrak{C} \wedge \mathfrak{D} \rightarrowtail \mathfrak{G} \wedge \mathfrak{H}$.*
6. *If $f \colon \mathfrak{C} \rightarrowtail \mathfrak{G}$ and $g \colon \mathfrak{G} \rightarrowtail \mathfrak{H}$ are skew fibrations, then so is $g \circ f \colon \mathfrak{C} \rightarrowtail \mathfrak{H}$.*

Proof. Straightforward. ◀

▶ **Construction 32.** *Let $\mathfrak{C}$ and $\mathfrak{D}$ be R&B-cographs such that $\mathfrak{C}^{\downarrow} = \mathfrak{G} \vee \mathfrak{H}$ and $\mathfrak{D}^{\downarrow} = \bar{\mathfrak{H}} \vee \mathfrak{K}$ for some cographs $\mathfrak{G}$, $\mathfrak{H}$, and $\mathfrak{K}$. We define the graph $\mathfrak{B} = \langle V_{\mathfrak{B}}, E_{\mathfrak{B}} \rangle$ with $V_{\mathfrak{B}} = V_{\mathfrak{G}} \uplus V_{\mathfrak{H}} \uplus V_{\mathfrak{K}}$*

and $E_{\mathfrak{B}} = B_{\mathfrak{C}} \uplus B_{\mathfrak{D}}$. *This allows us to define the R&B-cograph* $\mathfrak{E} = \mathfrak{C} \bullet \mathfrak{D}$ *as follows: We let* $\mathfrak{E}^{\downarrow} = \mathfrak{G} \vee \mathfrak{K}$, *i.e.,* $V_{\mathfrak{E}} = V_{\mathfrak{G}} \cup V_{\mathfrak{K}}$ *and* $R_{\mathfrak{E}} = E_{\mathfrak{G}} \cup E_{\mathfrak{K}}$, *and we let* $xy \in B_{\mathfrak{E}}$ *iff there is a path from* x *to* y *in* $\mathfrak{B}$. *Note that this indeed defines a perfect matching. For each* x *in* $V_{\mathfrak{E}}$ *there is a unique* y *connected to* x *by a path in* $\mathfrak{B}$ *because* $B_{\mathfrak{C}}$ *and* $B_{\mathfrak{D}}$ *are both perfect matchings.*

▶ **Lemma 33.** *If in Construction 32 the R&B-cographs* $\mathfrak{C}$ *and* $\mathfrak{D}$ *are critically chorded, then so is* $\mathfrak{E} = \mathfrak{C} \bullet \mathfrak{D}$.

Proof. This follows directly from the correspondence to MLL$^-$ proof nets given in [27] and the standard cut elimination result for linear logic proof nets. The idea used here goes back to [19], and a more recent presentation can be found in [15]. ◀

Next, we define for a simple flow $\phi\colon \Gamma \vdash B \wedge C$ the two *projections* $\phi_l\colon \Gamma \vdash B$ and $\phi_r\colon \Gamma \vdash C$ that are simple flows that "forget" the information about the deleted subformula. Their existence should not be surprising since from a proof of $B \wedge C$ one should be able to recover proofs of B and of C from the same premises.

▶ **Construction 34.** *Let* $\phi\colon \Gamma \vdash B \wedge C$ *be given by a critically chorded R&B-cograph* $\mathfrak{C}$ *and the skew fibration* $f\colon \mathfrak{C}^{\downarrow} \rightarrowtail \mathfrak{G}(\bar{\Gamma}) \vee (\mathfrak{G}(B) \wedge \mathfrak{G}(C))$. *Let* $U_C \subseteq V_{\mathfrak{C}}$ *be the set of all vertices in* $\mathfrak{C}$ *that are mapped by* f *to atom occurrences in* C, *and let* $U_C^{\perp} \subseteq V_{\mathfrak{C}}$ *be the smallest set such that*

- *If* $x \in U_C$ *and* $xy \in B_{\mathfrak{C}}$ *and* $y \notin U_C$ *then* $y \in U_C^{\perp}$.
- *If* $x \in U_C^{\perp}$ *and* $xy \in B_{\mathfrak{C}}$ *and* $y \notin U_C$ *then* $y \in U_C^{\perp}$.
- *If* $V', V'' \subseteq V_{\mathfrak{C}}$ *induce subcographs and* $V' \subseteq U_C^{\perp}$ *and* $V' \cap V'' = \varnothing$ *and* $V' \cup V''$ *induces a subcograph such that for all* $v' \in V'$ *and* $v'' \in V''$ *we have* $v'v'' \in R_{\mathfrak{C}}$, *then also* $V'' \subseteq U_C^{\perp}$.[3]

Now let $V_{\mathfrak{C}_l} = V \backslash (U_C \cup U_C^{\perp})$, *and let* $R_{\mathfrak{C}_l}$ *and* $B_{\mathfrak{C}_l}$ *be the restrictions of* $R_{\mathfrak{C}}$ *and* $B_{\mathfrak{C}}$ *(respectively) to* $V_{\mathfrak{C}_l}$. *Finally, we can define* $\phi_l\colon \Gamma \vdash B$ *by* $\mathfrak{C}_l = \langle V_{\mathfrak{C}_l}, R_{\mathfrak{C}_l}, B_{\mathfrak{C}_l} \rangle$ *and* $f_l\colon \mathfrak{C}_l^{\downarrow} \rightarrowtail \mathfrak{G}(\bar{\Gamma}) \vee \mathfrak{G}(B)$ *which is* f *restricted to* $V_{\mathfrak{C}_l}$.[4]

The idea behind this construction is to remove from $\mathfrak{C}$ the preimage of C together with the largest "critically chorded sub-(R&B-cograph)" that contains all vertices to which there is a B-edge from any vertex in the preimage of C.

It is easy to see that $\mathfrak{C}_l$ is critically chorded: any chordless æ-cycle would already be present in $\mathfrak{C}$, and any two vertices are connected by the same chordless æ-path as in $\mathfrak{C}$. We also have that $V_{\mathfrak{C}_l} \neq \varnothing$. To see that, let $U_B \subseteq V_{\mathfrak{C}}$ be the set of all vertices in $\mathfrak{C}$ that are mapped by f to atom occurrences in B. Now note that either both U_B and U_C are empty or both are not empty (because f is skew). If both U_B and U_C are empty, then $V_{\mathfrak{C}_l} = V$ which is not empty by definition. Otherwise, if U_B and U_C are both nonempty, but $V_{\mathfrak{C}_l}$ is, then all of U_B must be contained in $U_C^{\perp}$. Furthermore, at least one æ-paths connecting U_B and U_C starts and ends with a B-edge. Hence, an æ-cycle is closed by the R-edge between the two end vertices (because of the $\wedge$-connective between B and C), contradicting that $\mathfrak{C}$ is critically chorded.

Finally, it is easy to see that f_l is axiom preserving and a skew fibration. Thus, $\phi_l\colon \Gamma \vdash B$ is indeed a simple combinatorial flow. In the same way we can define the right projection

[3] This step can be seen as a combination of the $\downarrow$-, $\Downarrow$-, and $\uparrow$-steps in the empire construction in [2].

[4] As mentioned before, this construction is different from the one in [17], due to the absence of mix and "laxness". However, it remains open, how this compares to the "greedy garbage collection" of [17].

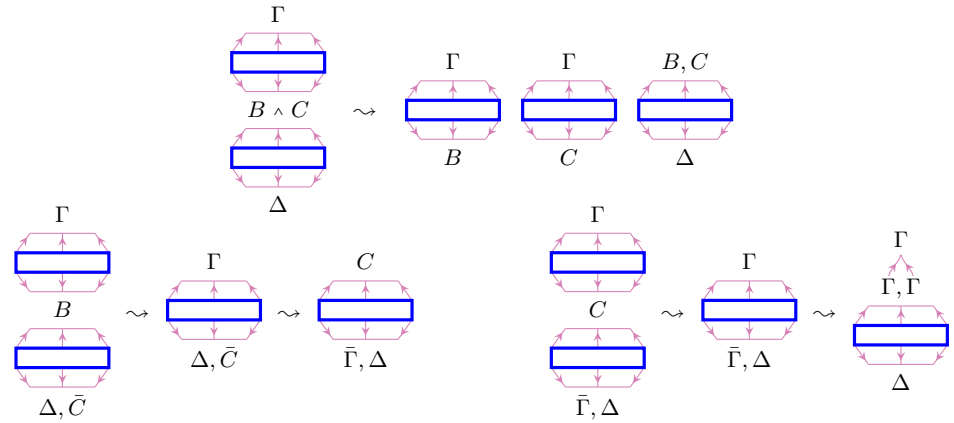
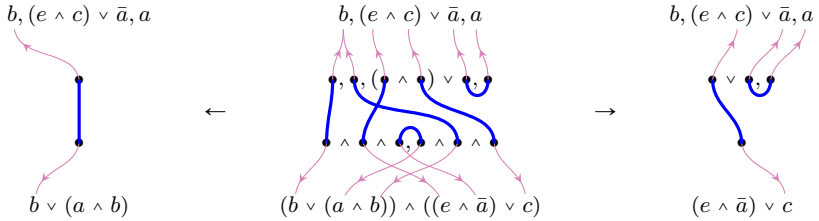


Figure 5 Steps in the proof of Lemma 30.

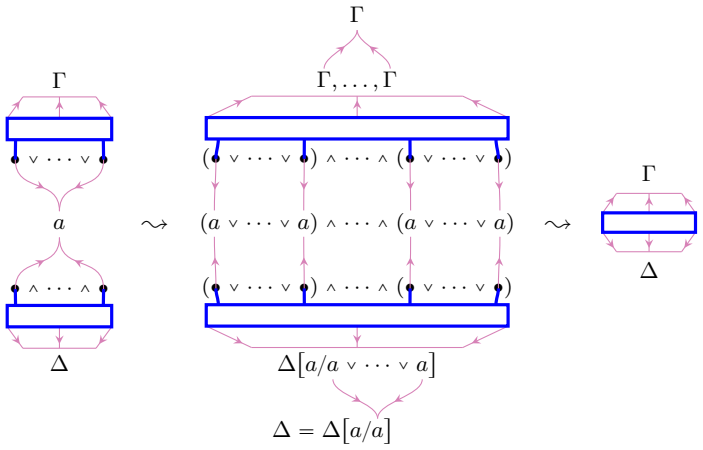
$\phi_r \colon \Gamma \vdash C$. Below is an example of a simple flow and its two projections:



(8)

In a dual way, we can define for a simple combinatorial flow $\psi \colon B \vee C \vdash \Delta$ its left and right projections $\psi_l \colon B \vdash \Delta$ and $\psi_r \colon C \vdash \Delta$.

Proof of Lemma 30. We proceed by induction on the formula A. First, assume $A = B \wedge C$. Then, from $\phi \colon \Gamma \vdash B \wedge C$ we can obtain the two projections $\phi_l \colon \Gamma \vdash B$ and $\phi_r \colon \Gamma \vdash C$, and from $\psi \colon B \wedge C \vdash \Delta$, we get $\psi' \colon B, C \vdash \Delta$ (see top line of Figure 5). From ψ' we can obtain (via Lemma 18) $\psi'' \colon B \vdash \Delta, \bar{C}$, which can be composed with ϕ_l to get, by induction hypothesis, a simple flow $\xi \colon \Gamma \vdash \Delta, \bar{C}$, from which (again by Lemma 18) we can get a simple flow $\chi' \colon C \vdash \bar{\Gamma}, \Delta$, as shown on the lower left of Figure 5. This can be composed with ϕ_r, which gives us by induction hypothesis a simple flow $\chi'' \colon \Gamma \vdash \bar{\Gamma}, \Delta$, from which we get a simple flow $\chi' \colon \Gamma, \Gamma \vdash \Delta$ by applying Lemma 18. Finally, we can apply Lemma 31 to get the desired $\chi \colon \Gamma \vdash \Delta$, as shown on the lower right Figure 5. If $A = B \vee C$ we proceed analogous. It remains to show the case when A is an atom, for which the construction is depicted below:



(9)

Here, let $f\colon \mathfrak{C}^{\downarrow} \rightarrowtail \mathfrak{G}(\bigwedge\bar{\Gamma}, a)$ and $g\colon \mathfrak{D}^{\downarrow} \rightarrowtail \mathfrak{G}(\bar{a}, \bigvee\Delta)$ be the skew fibrations of the simple flows $\phi\colon \Gamma \vdash a$ and $\psi\colon a \vdash \Delta$, respectively. Let $x_1, \ldots, x_n$ be the vertices in $\mathfrak{C}$ that are mapped by f to the a in the conclusion of ϕ, and let $y_1, \ldots, y_m$ be the vertices in $\mathfrak{D}$ that are mapped by g to the occurrence of $\bar{a}$ that represents the a in the premise of ψ.

Now we define the map $f^*\colon \mathfrak{C}^{\downarrow} \rightarrowtail \mathfrak{G}(\bigwedge\bar{\Gamma}, a \vee \cdots \vee a)$ where we replace a by a disjunction of n copies of a, and let f^* behave as f on $V_{\mathfrak{C}}\backslash\{x_1, \ldots, x_n\}$ and map each x_i to one copy of a. This clearly also is a skew fibration, and in a similar way we define the skew fibration $g^*\colon \mathfrak{D}^{\downarrow} \rightarrowtail \mathfrak{G}(\bar{a} \vee \cdots \vee \bar{a}, \bigvee\Delta)$ where we use m copies of $\bar{a}$. We let $\phi^*\colon \Gamma \vdash a \vee \cdots \vee a$ and $\psi^*\colon a \wedge \cdots \wedge a \vdash \Delta$ be the simple flows defined by f^* and g^*, respectively.

We then apply the construction of Section 4 to form the conjunction of m copies of ϕ^*, which yields a simple flow $\hat{\phi}\colon \Gamma, \ldots, \Gamma \vdash (a \vee \cdots \vee a) \wedge \cdots \wedge (a \vee \cdots \vee a)$.

Next, we substitute in ψ^* all simple flow paths that start in the premise $a \wedge \cdots \wedge a$ by the identity flow $\mathsf{id}\colon a \vee \cdots \vee a \vdash a \vee \cdots \vee a$ (with m copies of a on each side) as done in Section 5. Then we have a simple flow $\hat{\psi}\colon (a \vee \cdots \vee a) \wedge \cdots \wedge (a \vee \cdots \vee a) \vdash \Delta[a/a \vee \cdots \vee a]$.[5]

Finally, we plug $\hat{\phi}$ and $\hat{\psi}$ together and apply Lemma 33 to get a simple flow $\chi'\colon \Gamma, \ldots, \Gamma \vdash \Delta[a/a \vee \cdots \vee a]$, to which we apply Lemma 31 to get the desired simple flow $\chi\colon \Gamma \vdash \Delta$. ◀

7 Normalization IV: Putting things together

If we define the relation $\rightarrow$ on combinatorial flows such that $\phi_1 \rightarrow \phi_2$ whenever ϕ_1 can be reduced to ϕ_2 by one of the reductions given by Lemmas 24, 25, and 30, then we have immediately the following:

▶ **Theorem 35.** *The relation $\rightarrow$ is strongly normalizing, and the normal forms are simple combinatorial flows.*

Proof. At each step the number of simple combinatorial flows in the flow is reduced, and we always can make at least one reduction when the flow is not simple. ◀

▶ **Corollary 36.** *For each combinatorial flow $\phi\colon \Gamma \vdash \Delta$ there is a simple combinatorial flow $\phi'\colon \Gamma \vdash \Delta$ with the same premise and conclusion.*

8 Relation to to deep inference proofs

In this section we show how combinatorial flows are related to syntactic proofs in deep inference. It should be clear that the similar constructions are possible with other proof formalisms (like tableaux, sequent calculus, or resolution) as well.

We use the version of the deep inference system SKS [4], which is shown in Figure 6.[6] The rules shown there should be read as rewrite rule schemes that can be applied inside an arbitrary (positive) formula context. In the rules $\mathsf{ai}{\downarrow}$, $\mathsf{ai}{\uparrow}$, $\mathsf{ac}{\downarrow}$, and $\mathsf{ac}{\uparrow}$, the a can stand for any atom. In all rules, A, B, C, and D, can stand for any formula, and in $\mathsf{ai}{\downarrow}$ we additionally

[5] There is a slight abuse of notation: $\Delta[a/a \vee \cdots \vee a]$ stands for the sequent obtained by replacing every occurrence of a in Δ from which there is a simple flow path to an a in the premise of ψ^* by $a \vee \cdots \vee a$ (i.e., there might be occurrences of a in Δ that are not replaced).

[6] Note that our system is slightly different from the original version of SKS in [4]: We do not have explicit units in the language, and therefore our weakening rule is not atomic (see also [30]).

$$\mathsf{ai}{\downarrow}\ \frac{}{A \vee (a \wedge \bar{a})} \qquad \mathsf{s}\ \frac{(A \vee B) \wedge C}{A \vee (B \wedge C)} \qquad \mathsf{ai}{\uparrow}\ \frac{(\bar{a} \wedge a) \vee A}{A}$$

$$\mathsf{w}{\downarrow}\ \frac{A}{A \vee B} \qquad \mathsf{ac}{\downarrow}\ \frac{a \vee a}{a} \qquad \mathsf{m}\ \frac{(A \wedge C) \vee (B \wedge D)}{(A \vee B) \wedge (C \vee D)} \qquad \mathsf{ac}{\uparrow}\ \frac{a}{a \wedge a} \qquad \mathsf{w}{\uparrow}\ \frac{B \wedge A}{A}$$

Figure 6 Deep inference system SKS.

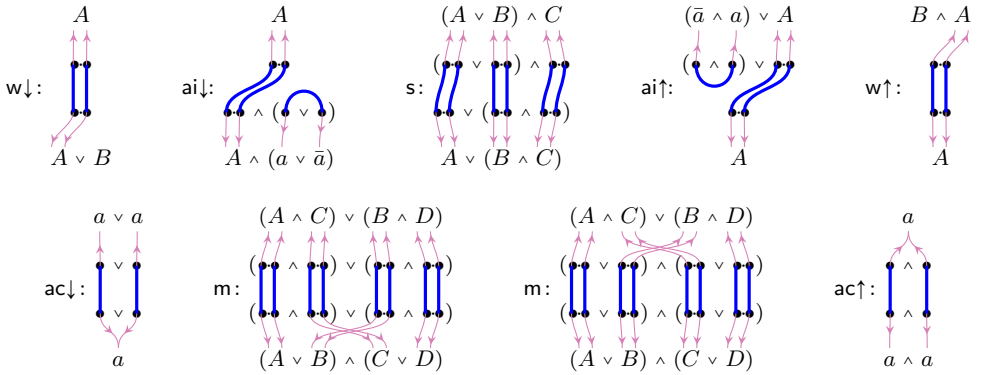


Figure 7 Simple combinatorial flows for the rules in Figure 6.

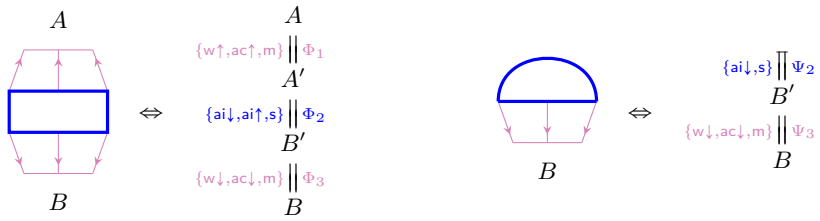


Figure 8 Relation between simple combinatorial flows and SKS derivations.

allow A to be empty, so to have proper proofs without premise.[7] We write

$$\begin{array}{c} P \\ \mathsf{s}\big\|\Phi \\ Q \end{array} \qquad \text{and} \qquad \begin{array}{c} \mathsf{s}\prod\Psi \\ Q \end{array} \tag{10}$$

to denote that there is a derivation Φ from P to Q, (respectively the proof Ψ without premise for the formula Q) in the system S, modulo the equivalence relation defined by associativity and commutativity of $\wedge$ and $\vee$, as given in (3). Figure 9 shows on the left an example of a derivation in SKS, where $2 \cdot \mathsf{s}$ stands for two consecutive applications of the s-rule.

Each rule in system SKS can straightforwardly be translated into a simple combinatorial flow, as indicated in Figure 7, where the double lines indicate the identity (see Observation 19). Note that for the m-rule there are two possible translations. Since whenever $A = B$ modulo associativity and commutativity (3) we have that $\mathfrak{G}(A) = \mathfrak{G}(B)$, an equivalence step in an

[7] We could also allow A to be empty in $\mathsf{ai}{\uparrow}$, so to have a proper refutation without conclusion.

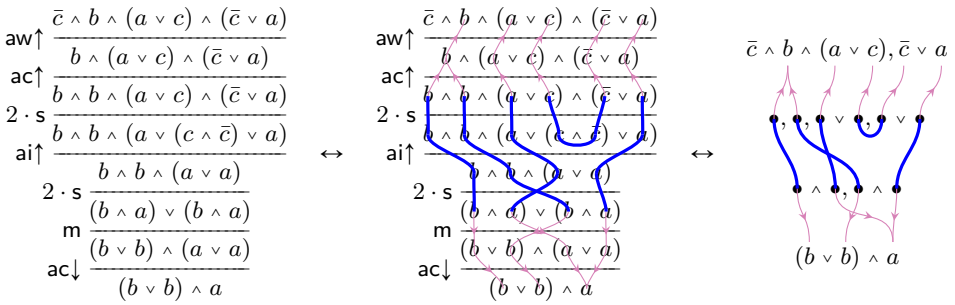


Figure 9 Example of an SKS derivation and its simple combinatorial flow.

SKS-proof can translated into the identity flow. This is enough to give a direct translation which proves the first direction of the following:

▶ **Theorem 37.** *Substitution-free combinatorial flows and system* SKS *p-simulate each other.*

For the other direction we use the relation between simple combinatorial flows and system SKS shown in Figure 8 to translate simple flows into SKS derivations which can be composed horizontally and vertically. Figure 9 shows the corresponding SKS derivations for the second example in Figure 1. For a detailed proof see [31].

Let us now investigate what happens when substitution is present. The substitution rule in a deductive system is given as follows:

$$\mathsf{sub}\,\frac{A}{A\sigma} \tag{11}$$

It replaces a formula A by the formula that is obtained by applying the substitution σ to A.

We define sSKS to be the system SKS + sub. It is important to note that unlike the other rules (shown in Figure 6) the rule sub in (11) cannot be applied inside a context. It is always applied to the whole formula. The reason is that the rule is not "strongly sound", in the sense that the premise does not imply the conclusion, as it is the case with the other inference rules. This means, in particular, that it does not make sense to speak of derivations in sSKS, but only of proofs with no premise. It has recently been established that sSKS is p-equivalent to Frege systems with substitution and Frege systems with extension [5, 30, 25]. Here we establish that combinatorial flows have the same expressiveness with respect to p-simulation:

▶ **Theorem 38.** *Combinatorial flows and* sSKS *p-simulate each other.*

The basic idea of the proof is to simulate the application of a substitution $\sigma = \{a_1 \mapsto B_1, \ldots, a_n \mapsto B_n\}$ in the sub-rule in sSKS by the substitution of the identity flow id_{B_i} for the variable a_i for each $i = 1..n$. But since in combinatorial flows the replacement is not performed simultaneously, we have to do a renaming first, in order to avoid unwanted variable capturing.

For the other direction, some more work is necessary. The reason is that in sSKS, substitution is a global rule, whereas in combinatorial flows it is a local activity, which is more flexible. To solve this problem, we use the notion of *extension*, due to [33], following the ideas used in [30]. For a detailed proof see [31].

9 Conclusion and Future Work

In this paper we proposed a solution to the problem of finding syntax-independent presentations of classical proofs that can also cover proof compression mechanisms that are usually studied in the area of proof complexity. This way, they can serve as a notion of *proof certificate* [23] that goes beyond mere cut-free sequent proofs.

Furthermore, the cut elimination presented in Section 6 can, together with the results of Section 8 also be used as an alternative normalization procedure for SKS derivations, since the normal forms are *streamlined* in the sense of [12] and [13].

The obvious next step is to include first-order quantifiers in the presentation. There is already preliminary work by Hughes [18] in this direction, but it still has to be investigated how the various notions of composition and normalization discussed in this paper behave in the presence of quantifiers.

Another direction of possible future research is the question whether combinatorial flows can form some free category (in the same sense as MLL proof nets form the free unit-free star-autonomous category [14]) and the relation to categorical combinators [7].

Acknowledgements. I thank Anupam Das and Alessio Guglielmi for fruitful discussions, and I thank Paola Bruscoli, Dominic Hughes, and anonymous referees for helpful comments on earlier drafts of this work.

References

1 Peter B. Andrews. Refutations by matings. *IEEE Transactions on Computers*, C-25:801–807, 1976.

2 Gianluigi Bellin and Jacques van de Wiele. Subnets of proof-nets in MLL$^-$. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Notes*, pages 249–270. Cambridge University Press, 1995.

3 Wolfgang Bibel. On matrices with connections. *Journal of the ACM*, 28:633–645, 1981.

4 Kai Brünnler and Alwen Fernanto Tiu. A local system for classical logic. In R. Nieuwenhuis and A. Voronkov, editors, *LPAR 2001*, volume 2250 of *LNAI*, pages 347–361. Springer, 2001.

5 Paola Bruscoli and Alessio Guglielmi. On the proof complexity of deep inference. *ACM Transactions on Computational Logic*, 10(2):1–34, 2009. Article 14.

6 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979.

7 Pierre-Louis Curien. Categorical combinators. *Information and Control*, 69(1-3):188–254, 1986. doi:10.1016/S0019-9958(86)80047-X.

8 Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Annals of Mathematical Logic*, 28:181–203, 1989.

9 Anupam Das. Rewriting with linear inferences in propositional logic. In Femke van Raamsdonk, editor, *24th International Conference on Rewriting Techniques and Applications (RTA)*, volume 21 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 158–173. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013.

10 R. J. Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10(2):303–318, 1965.

11 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

12 Alessio Guglielmi and Tom Gundersen. Normalisation control in deep inference via atomic flows. *Logical Methods in Computer Science*, 4(1:9):1–36, 2008. URL: http://arxiv.org/abs/0709.1205.

13 Alessio Guglielmi, Tom Gundersen, and Lutz Straßburger. Breaking paths in atomic flows for classical logic. In *LICS 2010*, 2010.

14 Willem Heijltjes and Lutz Straßburger. Proof nets and semi-star-autonomous categories. *Mathematical Structures in Computer Science*, 26(5):789–828, 2016. doi:10.1017/S0960129514000395.

15 Dominic Hughes. Simple multiplicative proof nets with units. Preprint, 2005. URL: http://arxiv.org/abs/math.CT/0507003.

16 Dominic Hughes. Proofs Without Syntax. *Annals of Mathematics*, 164(3):1065–1076, 2006.

17 Dominic Hughes. Towards Hilbert's 24^{th} problem: Combinatorial proof invariants: (preliminary version). *Electr. Notes Theor. Comput. Sci.*, 165:37–63, 2006.

18 Dominic Hughes. First-order proofs without syntax. Berkeley Logic Colloquium, 2014.

19 Gregory Maxwell Kelly and Saunders Mac Lane. Coherence in closed categories. *J. of Pure and Applied Algebra*, 1:97–140, 1971.

20 Jan Krajíček and Pavel Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *The Journal of Symbolic Logic*, 54(3):1063–1079, 1989.

21 François Lamarche and Lutz Straßburger. Naming proofs in classical propositional logic. In Paweł Urzyczyn, editor, *TLCA'05*, volume 3461 of *LNCS*, pages 246–261. Springer, 2005. URL: http://www.lix.polytechnique.fr/~lutz/papers/namingproofsCL.pdf.

22 Olivier Laurent. Polarized proof-nets: proof-nets for LC (extended abstract). In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications (TLCA 1999)*, volume 1581 of *LNCS*, pages 213–227. Springer, 1999.

23 Dale Miller. A proposal for broad spectrum proof certificates. In J.-P. Jouannaud and Z. Shao, editors, *CPP: First International Conference on Certified Programs and Proofs*, volume 7086 of *Lecture Notes in Computer Science*, pages 54–69, 2011.

24 Rolf H. Möhring. Computationally tractable classes of ordered sets. In I. Rival, editor, *Algorithms and Order*, pages 105–194. Kluwer Acad. Publ., 1989.

25 Novak Novakovic and Lutz Straßburger. On the power of substitution in the calculus of structures. *ACM Trans. Comput. Log.*, 16(3):19, 2015.

26 Christian Retoré. *Réseaux et Séquents Ordonnés*. PhD thesis, Université Paris VII, 1993.

27 Christian Retoré. Handsome proof-nets: perfect matchings and cographs. *Theoretical Computer Science*, 294(3):473–488, 2003.

28 Edmund P. Robinson. Proof nets for classical logic. *Journal of Logic and Computation*, 13:777–797, 2003.

29 Lutz Straßburger. From deep inference to proof nets via cut elimination. *Journal of Logic and Computation*, 21(4):589–624, 2011.

30 Lutz Straßburger. Extension without cut. *Annals of Pure and Applied Logic*, 163(12):1995–2007, 2012.

31 Lutz Straßburger. Combinatorial Flows and Proof Compression. Research Report RR-9048, Inria Saclay, 2017. URL: https://hal.inria.fr/hal-01498468.

32 Anne Sjerp Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, second edition, 2000.

33 G. S. Tseitin. On the complexity of derivation in propositional calculus. *Zapiski Nauchnykh Seminarou LOMI*, 8:234–259, 1968.

Streett Automata Model Checking of Higher-Order Recursion Schemes*

Ryota Suzuki[1], Koichi Fujima[2], Naoki Kobayashi[3], and Takeshi Tsukada[4]

1 **The University of Tokyo, Tokyo, Japan**
rsuzuki@kb.is.s.u-tokyo.ac.jp
2 **The University of Tokyo, Tokyo, Japan**
kfujima@kb.is.s.u-tokyo.ac.jp
3 **The University of Tokyo, Tokyo, Japan**
koba@kb.is.s.u-tokyo.ac.jp
4 **The University of Tokyo, Tokyo, Japan**
tsukada@kb.is.s.u-tokyo.ac.jp

Abstract

We propose a practical algorithm for Streett automata model checking of higher-order recursion schemes (HORS), which checks whether the tree generated by a given HORS is accepted by a given Streett automaton. The Streett automata model checking of HORS is useful in the context of liveness verification of higher-order functional programs. The previous approach to Streett automata model checking converted Streett automata to parity automata and then invoked a parity tree automata model checker. We show through experiments that our direct approach outperforms the previous approach. Besides being able to directly deal with Streett automata, our algorithm is the first practical Streett or parity automata model checking algorithm that runs in time polynomial in the size of HORS, assuming that the other parameters are fixed. Previous practical fixed-parameter polynomial time algorithms for HORS could only deal with the class of trivial tree automata. We have confirmed through experiments that (a parity automata version of) our model checker outperforms previous parity automata model checkers for HORS.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Higher-order model checking, higher-order recursion schemes, Streett automata

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.32

1 Introduction

Model checking of higher-order recursion schemes (HORS, for short) [13, 25] has been actively studied and applied to automated verification of higher-order programs [17, 21, 24, 26, 34, 23, 41]. The model checking problem asks whether the tree generated by a given HORS is accepted by a given tree automaton. Despite the extremely high complexity (k-EXPTIME complete for order-k HORS), practical model checkers that work reasonably well for typical inputs have been developed [14, 16, 3, 31, 29, 7, 28]. In particular, the state-of-the-art trivial automata model checkers for HORS (i.e., model checkers which handle the restricted class of automata called trivial tree automata [1]) [3, 18, 31] can handle thousands of lines of input in a few seconds. The state-of-the-art model checkers for the full class of tree automata (that

* This work was supported by JSPS Kakenhi 15H05706.

is equi-expressive to the modal μ-calculus and MSO) have, however, been much behind the trivial automata model checkers. Indeed, whilst the state-of-the-art trivial automata model checkers for HORS employ fixed-parameter polynomial time algorithms, existing parity tree automata model checkers for HORS [7, 28] do not. Another limitation of the state-of-the-art model checkers for HORS is that the class of automata is restricted to trivial or parity tree automata. Whilst parity tree automata are equi-expressive to other classes of tree automata like Streett, Rabin, and Muller automata, the translation from those automata to parity tree automata significantly increases the number of states. Thus, it may be desirable for model checkers to support other classes of automata directly.

To address the limitations above, we propose a practical Streett automata model checking algorithm for HORS, which checks whether the tree generated by a given HORS is accepted by a given Streett automaton. Compared with the previous model checking algorithms for HORS that can deal with the full class of tree automata [7, 28], our new algorithm has the following advantages: (i) It can directly deal with Streett automata, which naturally arise in the context of liveness verification of higher-order programs [41]. (ii) More importantly, it runs in time polynomial in the size of HORS, assuming that the other parameters (the size of the automaton and the largest order and arity of non-terminals in HORS) are fixed. The previous parity automata model checkers for HORS [7, 28] did not satisfy this property, and suffered from hyper-exponential time complexity in the size of HORS.

We develop the algorithm in two steps. First, following Kobayashi and Ong's type system for parity automata model checking of HORS [19], we prepare a type system for Streett automata model checking such that the tree generated by a HORS $\mathcal{G}$ is accepted by a Streett automaton $\mathcal{A}$ if and only if $\mathcal{G}$ is typable in the type system parameterized by $\mathcal{A}$. We prove its correctness by showing that the type system can actually be viewed as an instance of Tsukada and Ong's type system [38]. Secondly, we develop a practical algorithm for checking the typability. The algorithm has been inspired by Broadbent and Kobayashi's saturation-based algorithm [3] for trivial automata model checking; in fact, the algorithm is a simple modification of their HorSatT algorithm. The proof of the correctness of our algorithm is, however, non-trivial and much more involved than the correctness proof for HorSatT. The correctness proof is one of the main contributions of the present paper.

We have implemented a new model checker HorSatS based on the proposed algorithm and its variation, called HorSatP,[1] for parity tree automata model checking, and experimentally confirmed the two advantages above. For the advantage (i), we have confirmed that HorSatS is often faster than the combination of a converter from Streett to parity tree automata, and HorSatP. For (ii), we have confirmed that HorSatP often outperforms previous parity automata model checkers [7, 28].

The rest of the paper is organized as follows. Section 2 reviews basic definitions. Section 3 provides a type-based characterization of Streett automata model checking of HORS and proves its correctness. Section 4 develops a practical algorithm for Streett automata model checking and proves its correctness. Section 5 reports experimental results. Section 6 discusses related work and Section 7 concludes the paper. Proofs omitted in this paper are found in a longer version of the paper [36].

[1] Actually, HorSatP has been implemented in 2015 and used in Watanabe et al.'s work [41]. It has not been properly formalized, however.

2 Preliminaries

In this section we review the definitions of higher-order recursion schemes (HORS) [12, 25, 17] and (alternating) Streett tree automata [35, 8] to define Streett automata model checking of HORS. We write $dom(f)$ for the domain of a map f, and $\widetilde{x}$ for a sequence $x_1, x_2, \ldots, x_m$ (for some m). Given a set Σ of symbols (which we call *terminals*), a Σ-*labeled (unranked) tree* is a partial map T from $\{1, \ldots, N\}^*$ (for some fixed $N \in \mathbb{N}$) to Σ such that if $\pi i \in dom(T)$, then $\pi \in dom(T)$ and $\pi j \in dom(T)$ for all $1 \leq j \leq i$. If Σ is a *ranked alphabet* (i.e., a map from terminals to natural numbers), a Σ-*labeled ranked tree* T is a $dom(\Sigma)$-labeled tree such that for each $\pi \in dom(T)$, $\{i \mid \pi i \in dom(T)\} = \{1, \ldots, \Sigma(T(\pi))\}$.

2.1 Higher-Order Recursion Schemes

The set of *sorts*, ranged over by κ, is defined by $\kappa ::= \circ \mid \kappa_1 \to \kappa_2$. We sometimes abbreviate $\underbrace{\kappa \to \cdots \kappa}_{n} \to \kappa'$ to $\kappa^n \to \kappa'$. The sorts can be viewed as the types of the simply-typed λ-calculus with the single base type $\circ$, which is the type of trees. We write $\mathbf{Terms}_{\mathcal{K},\kappa}$ for the set of simply-typed λ-terms that have the sort κ under the environment $\mathcal{K}$. The *order* and *arity* of each sort, denoted by $ord(\kappa)$ and $arity(\kappa)$, are defined inductively by: $ord(\circ) = arity(\circ) = 0$, $ord(\kappa_1 \to \kappa_2) = \max(ord(\kappa_1) + 1, ord(\kappa_2))$, and $arity(\kappa_1 \to \kappa_2) = arity(\kappa_2) + 1$.

▶ **Definition 1** (higher-order recursion scheme)**.** A *higher-order recursion scheme* (HORS) is a quadruple $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ where: (i) Σ is a ranked alphabet. (ii) $\mathcal{N}$ is a map from symbols called *non-terminals* to sorts. (iii) $\mathcal{R}$ is a map from non-terminals to simply-typed terms of the form $\lambda\widetilde{x}.t$, where t does not include λ-abstractions and has the sort $\circ$. It is required that for each $F \in dom(\mathcal{N})$, $\lambda\widetilde{x}.t \in \mathbf{Terms}_{\mathcal{N} \cup \{a : \circ^{\Sigma(a)} \to \circ \mid a \in dom(\Sigma)\}, \mathcal{N}(F)}$.
$S \in dom(\mathcal{N})$ is a special non-terminal such that $\mathcal{N}(S) = \circ$.

The *rewriting relation* $\longrightarrow_{\mathcal{G}}$ on terms of $\mathcal{G}$ is defined inductively by: (i) $F\ s_1 \ldots s_m \longrightarrow_{\mathcal{G}} [s_1/x_1, \ldots, s_m/x_m]t$ if $\mathcal{R}(F) = \lambda x_1 \ldots x_m.t$, (ii) $s\ t \longrightarrow_{\mathcal{G}} s'\ t$ if $s \longrightarrow_{\mathcal{G}} s'$, and (iii) $s\ t \longrightarrow_{\mathcal{G}} s\ t'$ if $t \longrightarrow_{\mathcal{G}} t'$. Here, $[s_1/x_1, \ldots, s_m/x_m]t$ is the term obtained from t by replacing each x_i with s_i. The tree $[\![\mathcal{G}]\!]$ generated by $\mathcal{G}$, called the *value tree* of $\mathcal{G}$, is defined as the least upper bound of $\{t^\perp \mid S \longrightarrow_{\mathcal{G}}^* t\}$ with respect to $\sqsubseteq$ where $t^\perp$ is defined by (i) $t^\perp = t$ if t is a terminal, (ii) $(t_1\ t_2)^\perp = t_1^\perp\ t_2^\perp$ if $t_1^\perp \neq \perp$, and (iii) $t^\perp = \perp$ otherwise. Here, the partial order $\sqsubseteq$ is defined by $t_1 \sqsubseteq t_2$ if t_2 is obtained by replacing some of $\perp$'s in t_1 with some trees. The value tree $[\![\mathcal{G}]\!]$ is a $(\Sigma \cup \{\perp \mapsto 0\})$-labeled ranked tree.

▶ **Example 2** (HORS)**.** Let $\mathcal{G}_1 = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ where $\Sigma = \{\mathtt{a} \mapsto 2,\ \mathtt{b} \mapsto 1,\ \mathtt{c} \mapsto 0\}$, $\mathcal{N} = \{F \mapsto ((\circ \to \circ) \to \circ),\ B \mapsto ((\circ \to \circ) \to \circ \to \circ),\ I \mapsto (\circ \to \circ),\ S \mapsto \circ\}$, and $\mathcal{R} = \{F \mapsto (\lambda f.\ f\ (\mathtt{a}\ (f\ \mathtt{c})\ (F\ (B\ f)))),\ B \mapsto (\lambda f x.\ \mathtt{b}\ (f\ x)),\ I \mapsto (\lambda x.\ x),\ S \mapsto (F\ I)\}$. From the start non-terminal S, the reduction proceeds as follows.

$$S \longrightarrow_{\mathcal{G}_1} F\ I \longrightarrow_{\mathcal{G}_1} I\ (\mathtt{a}\ (I\ \mathtt{c})\ (F\ (B\ I))) \longrightarrow_{\mathcal{G}_1}^* \mathtt{a}\ \mathtt{c}\ (F\ (B\ I))$$
$$\longrightarrow_{\mathcal{G}_1}^* \mathtt{a}\ \mathtt{c}\ (\mathtt{b}\ (\mathtt{a}\ (\mathtt{b}\ \mathtt{c})\ (F\ (B\ (B\ I))))) \longrightarrow_{\mathcal{G}_1} \ldots$$

The value tree $[\![\mathcal{G}_1]\!]$ has a finite path $\mathtt{abab}^2 \ldots \mathtt{ab}^n\mathtt{ab}^n\mathtt{c}$ for each $n \in \mathbb{N}$ and also has an infinite path $\mathtt{abab}^2\mathtt{ab}^3 \ldots$.

2.2 Streett Tree Automata

Given a set X, the set $\mathcal{B}^+(X)$ of *positive boolean formulas* over X, ranged over by φ, is defined by $\varphi ::= \mathbf{true} \mid \mathbf{false} \mid x \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$ where x ranges over X. Given a subset

Y of X, one can calculate the boolean value $[\![\varphi]\!]_Y$ of φ by assigning **true** to the elements of Y and **false** to those of $X \setminus Y$. We say Y *satisfies* φ if $[\![\varphi]\!]_Y = \mathbf{true}$.

▶ **Definition 3** (Streett tree automaton [35, 8])**.** A *Streett tree automaton* is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, \mathcal{C})$ where Σ is a ranked alphabet, Q is a finite set of *states*, δ is a map from $Q \times dom(\Sigma)$ to $\mathcal{B}^+(\mathbb{N} \times Q)$ called a *transition function* such that $\delta(q, a) \in \mathcal{B}^+(\{1, \ldots, \Sigma(a)\} \times Q)$ for each q and a, and $q_0 \in Q$ is a special state called the *initial state*, and $\mathcal{C}$ is a *Streett acceptance condition* of the form $\mathcal{C} = \{(E_1, F_1), \ldots, (E_k, F_k)\}$ where $E_i, F_i \subseteq Q$ for each i. Given a Σ-labeled ranked tree T, a *run-tree* of $\mathcal{A}$ over T is a $(dom(T) \times Q)$-labeled (unranked) tree R such that (i) $\varepsilon \in dom(R)$, (ii) $R(\varepsilon) = (\varepsilon, q_0)$, (iii) for every $\pi \in dom(R)$ with $R(\pi) = (\xi, q)$ and $j \in \{j \in \mathbb{N} \mid \pi j \in dom(R)\}$, there exist $i \in \mathbb{N}$ and $q' \in Q$ such that $R(\pi j) = (\xi i, q')$, and (iv) for every $\pi \in dom(R)$ with $R(\pi) = (\xi, q)$, $\{(i, q') \mid {}^{\exists} j.\ R(\pi j) = (\xi i, q')\}$ satisfies $\delta(q, T(\xi))$. Let $Paths(R)$ be defined by $Paths(R) = \{\pi \in \mathbb{N}^\omega \mid \text{every (finite) prefix of } \pi \text{ is in } dom(R)\}$ and $Inf_R : Paths(R) \to 2^Q$ be defined by $Inf_R(\pi) = \{q \in Q \mid \text{the state label of } R(\pi_i) \text{ is } q \text{ for infinitely many } i\}$ where π_i is the prefix of π of length i. A run-tree R is *accepting* if for every $\pi \in Paths(R)$ and every $(E_i, F_i) \in \mathcal{C}$, $(Inf_R(\pi) \cap E_i \neq \emptyset \Rightarrow Inf_R(\pi) \cap F_i \neq \emptyset)$ holds. A Streett automaton $\mathcal{A}$ *accepts* a Σ-labeled ranked tree T if there exists an accepting run-tree of $\mathcal{A}$ over T.

▶ **Example 4** (Streett tree automaton)**.** Let $\mathcal{A}_1 = (\Sigma, Q, \delta, q_0, \mathcal{C})$ be a Streett tree automaton where $\Sigma = \{\mathtt{a} \mapsto 2,\ \mathtt{b} \mapsto 1,\ \mathtt{c} \mapsto 0\}$, $Q = \{q_0, q_\mathtt{a}, q_\mathtt{b}\}$, $\delta(q, \mathtt{a}) = (1, q_\mathtt{a}) \wedge (2, q_\mathtt{a})$ for each q, $\delta(q, \mathtt{b}) = (1, q_\mathtt{b})$ for each q, and $\delta(q, \mathtt{c}) = \mathbf{true}$ for each q, and $\mathcal{C} = \{(E_1, F_1)\}$ where $E_1 = \{q_\mathtt{a}\}$ and $F_1 = \{q_\mathtt{b}\}$. This automaton accepts Σ-labeled ranked trees such that for each infinite path, if $\mathtt{a}$ appears infinitely often in it, then $\mathtt{b}$ also appears infinitely often in it.

2.3 Streett Automata Model Checking Problem for HORS

We can now define the *Streett automata model checking problem for HORS*.

▶ **Definition 5** (Streett automata model checking problem for HORS)**.** The Streett model checking problem for HORS is a decision problem to check whether $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$, for a given HORS $\mathcal{G}$ and a Streett tree automaton $\mathcal{A}$.

The need for Streett automata model checking of HORS naturally arises in the context of verifying liveness properties of higher-order programs. A popular method for verification of temporal properties of programs is to use Vardi's reduction to fair termination [40], the problem of checking whether all the *fair* execution sequences of a given program are terminating [5, 2, 27, 41]. Here, a fairness constraint is of the form $\{(A_1, B_1), \ldots, (A_n, B_n)\}$, which means that if the event A_i occurs infinitely often, so does B_i, for each i. For proving/disproving fair termination of a higher-order functional program, a natural approach is to convert the program to a HORS that generates a tree representing all the possible event sequences, and then check whether the tree contains a fair but infinite event sequence. For example, consider the program:

$$\textbf{let rec } f() = \textbf{if } *_{\mathrm{int}} < 0 \textbf{ then } (\textbf{event } B;\ ()) \textbf{ else } (\textbf{event } A;\ f())$$

where $*_{\mathrm{int}}$ represents a non-deterministic integer. It can be converted to the following HORS, whose value tree represents all the possible event sequences.

$$S \to F \qquad F \to \mathtt{br}\ (\mathtt{ev}_B\ \mathtt{end})\ (\mathtt{ev}_A\ F).$$

Then, the problem of checking that the original program is not terminating with respect to the fairness constraint $\{(A, B)\}$ is reduced to the problem of checking that the tree

generated by the HORS has a fair infinite path (i.e., an infinite path such that if $\mathtt{ev}_A$ occurs infinitely often, so does $\mathtt{ev}_B$). In the case of the HORS above, there is no infinite path in which $\mathtt{ev}_B$ occurs infinitely often, from which we can conclude that the original program is fair terminating. Indeed, Watanabe et al. [41] took such an approach for disproving fair termination of functional programs.[2] The resulting decision problem for HORS can naturally be expressed as a Streett model checking problem; in the above case, we can use the Streett automaton $\mathcal{A} = (\{\mathtt{br} \mapsto 2, \mathtt{ev}_A \mapsto 1, \mathtt{ev}_B \mapsto 1, \mathtt{end} \mapsto 0\}, \{q_0, q_A, q_B\}, \delta, q_0, \{(\{q_A\}, \{q_B\})\})$ where $\delta(q, \mathtt{ev}_A) = (1, q_A)$, $\delta(q, \mathtt{ev}_B) = (1, q_B)$, $\delta(q, \mathtt{br}) = (1, q_0) \wedge (2, q_0)$, and $\delta(q, \mathtt{end}) = \mathbf{true}$ for every $q \in \{q_0, q_A, q_B\}$.

As usual [25, 19], we assume that the value tree $[\![\mathcal{G}]\!]$ of a HORS $\mathcal{G}$ does not contain $\bot$ in the rest of the paper. Note that this is not a limitation, because any instance of the model checking problem for a HORS $\mathcal{G}$ and a Streett automaton $\mathcal{A}$ can be reduced to an equivalent one for $\mathcal{G}'$ and $\mathcal{A}'$ such that $[\![\mathcal{G}']\!]$ does not contain $\bot$.

3 A Type System for Streett Automata Model Checking

This section presents an intersection type system (parameterized by a Streett automaton $\mathcal{A}$) for Streett automata model checking of HORS, such that a HORS $\mathcal{G}$ is typable in the type system if and only if $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$. The type system is obtained by modifying the Kobayashi-Ong type system [19] for parity automata model checking. We prove the correctness of our type system by showing that it is actually an instance of Tsukada and Ong's type system for model checking Böhm trees [38].

Let $\mathcal{A} = (\Sigma, Q, \delta, q_0, \mathcal{C})$ be a Streett automaton with $\mathcal{C} = \{(E_1, F_1), \ldots, (E_k, F_k)\}$. We define the set of *effects* by $\mathbb{E} = 2^{\{E_1, \ldots, E_k, F_1, \ldots, F_k\}}$.[3] Here, $E_1, \ldots, E_k, F_1, \ldots, F_k$ in effects should be considered just symbols (so that they are different from each other, even if E_i and F_j in $\mathcal{C}$ happen to be the same set of states) although they intuitively represent the sets used in $\mathcal{C}$. The set of *prime types*, ranged over by θ, and the set of *intersection types*, ranged over by τ, are defined by:

$$\theta \text{ (prime types)} ::= q \mid \tau \to \theta' \qquad \tau \text{ (intersection types)} ::= \{(\theta_i, e_i)\}_{i \in I}$$

where $q \in Q$, $e_i \in \mathbb{E}$ and I is a finite index set. Note that $\{(\theta_i, e_i)\}_{i \in I}$ is a shorthand for $\{(\theta_i, e_i) \mid i \in I\}$. We sometimes write $(\theta_1, e_1) \wedge \cdots \wedge (\theta_k, e_k)$ for $\{(\theta_1, e_1), \ldots, (\theta_k, e_k)\}$, and $\top$ for the empty intersection type $\emptyset$.

Intuitively, q is the type of trees accepted from q by $\mathcal{A}$ (i.e., by $\mathcal{A}_q = (\Sigma, Q, \delta, q, \mathcal{C})$), and $\{(\theta_i, e_i)\}_{i \in I} \to \theta'$ is the type of functions which take an argument that has type θ_i for every $i \in I$, and return a value of type θ'. Here, e_i describes what states may/must be visited before the argument is used as a value of type θ_i. For example, the type $(q_1, \{E_1\}) \wedge (q_2, \{F_1\}) \to q$ describes the type of functions that take a tree that can be accepted from both q_1 and q_2 as an argument, and return a tree of type q. Furthermore, the effect parts ($\{E_1\}$ and $\{F_1\}$) describe that in an accepting run of $\mathcal{A}_q$ over the returned tree, the argument can be used as a value of type q_1 (i.e., visited with state q_1) only after visiting states in E_1, and also used as a value of type q_2 only after visiting states in F_1.

[2] The actual method in [41] is more complicated due to a combination with predicate abstraction.

[3] One can use $\mathbb{E} = 2^{\{E_1, \ldots, E_k, F_1, \ldots, F_k\}}/\sim$ instead, where $\sim$ is an equivalence relation defined in the proof of Theorem 8. This improves the time complexity of our algorithm. We use $\mathbb{E} = 2^{\{E_1, \ldots, E_k, F_1, \ldots, F_k\}}$ here for understandability, however.

$$\frac{}{\{x : (\theta, e_{\mathbf{id}})\} \vdash_{\mathcal{A}} x : \theta} \quad \text{(Var)}$$

$$\frac{q \in Q \qquad a \in \Sigma \qquad n = \Sigma(a) \qquad \{(i, q_{ij}) \mid i \in \{1, \ldots, n\}, j \in J_i\} \text{ satisfies } \delta(q, a) \qquad e_{ij} = \mathit{Eff}(q_{ij})}{\emptyset \vdash_{\mathcal{A}} a : \{(q_{1j}, e_{1j})\}_{j \in J_1} \to \cdots \to \{(q_{nj}, e_{nj})\}_{j \in J_n} \to q} \quad \text{(Const)}$$

$$\frac{\Gamma_0 \vdash_{\mathcal{A}} t_0 : \{(\theta_i, e_i)\}_{i \in I} \to \theta \qquad \Gamma_i \vdash_{\mathcal{A}} t_1 : \theta_i \text{ for each } i \in I}{\Gamma_0 \cup \bigcup_{i \in I} (\Gamma_i \Uparrow e_i) \vdash_{\mathcal{A}} t_0\ t_1 : \theta} \quad \text{(App)}$$

$$\frac{\Gamma \cup \{x : (\theta_i, e_i) \mid i \in I\} \vdash_{\mathcal{A}} t : \theta \qquad \Gamma \text{ has no bindings for } x \qquad I \subseteq J}{\Gamma \vdash_{\mathcal{A}} \lambda x.t : \{(\theta_i, e_i)\}_{i \in J} \to \theta} \quad \text{(Abs)}$$

Figure 1 Typing Rules.

We say that a prime type θ is a *refinement* of a sort κ, written $\theta :: \kappa$, when it is derivable from the following rules: (i) For each $q \in Q$, $q :: \circ$, and (ii) $(\{(\theta_i, e_i)\}_{i \in I} \to \theta) :: (\kappa_0 \to \kappa_1)$ if $\theta_i :: \kappa_0$ for all $i \in I$ and $\theta :: \kappa_1$. We say a prime type θ is *well-formed* if $\theta :: \kappa$ for some κ. We consider only well-formed prime types below.

A *type environment* is a set of type bindings of the form $x : (\theta, e)$ where x is a variable or a non-terminal, θ is a prime type, and e is an effect. The part e represents *when* x may be used as a value of type θ. Note that a type environment may contain multiple bindings for each variable.

The *type judgement relation* $\vdash_{\mathcal{A}}$ among type environments, terms and prime types are defined inductively by the typing rules in Figure 1. The operations used in the rules are defined by: $e_{\mathbf{id}} = \emptyset$, $\mathit{Eff}(q) = \{E \in \{E_1, \ldots, E_k, F_1, \ldots, F_k\} \mid q \in E\}$, and $\Gamma \Uparrow e = \{x : (\theta, e \circ e') \mid (x : (\theta, e')) \in \Gamma\}$ where $e \circ e' = e \cup e'$.

In the rule (Var), the effect $e_{\mathbf{id}}$ indicates that no state has been visited before the use of x. The rule (Const) is for terminals (i.e., tree constructors); the premise "$\{(i, q_{ij}) \mid i \in \{1, \ldots, n\}, j \in J_i\}$ satisfies $\delta(q, a)$" implies that a tree $a\, T_1 \cdots T_n$ is accepted from q if T_i is accepted from q_{ij} for each $j \in J_i$, hence the type of a in the conclusion. In addition, the effect $e_{ij} = \mathit{Eff}(q_{ij})$ reflects the fact that the state q_{ij} has been visited. In the rule (App), each type environment Γ_i is "lifted" by e_i, to reflect the condition (as indicated by the argument type of t_0) that the argument t_1 is used as a value of type θ_i only after the effect e_i occurs. In the rule (Abs) for λ-abstractions, we allow weakening on the types of x.

▶ **Example 6** (type judgement). Let $\mathcal{A}_1 = (\Sigma, Q, \delta, q_0, \mathcal{C})$ where $\Sigma = \{\mathtt{a} \mapsto 2,\ \mathtt{b} \mapsto 1\}$, $Q = \{q_0, q_\mathtt{a}, q_\mathtt{b}\}$, $\mathcal{C} = \{(E_1, F_1)\}$ with $E_1 = \{q_\mathtt{a}\}$ and $F_1 = \{q_\mathtt{b}\}$, and $\delta(q, \mathtt{a}) = (1, q_\mathtt{a}) \wedge (2, q_\mathtt{a})$ for each q and $\delta(q, \mathtt{b}) = (1, q_\mathtt{b})$ for each q. Types of the terminals can be determined by the (Const) rule; for example, one can derive $\emptyset \vdash_{\mathcal{A}} \mathtt{a} : (q_\mathtt{a}, \{E_1\}) \to (q_\mathtt{a}, \{E_1\}) \to q_0$. By using the typing rules, one can derive a type judgement: $\{x : (q_\mathtt{a}, \{E_1\}),\ x : (q_\mathtt{b}, \{F_1\})\} \vdash_{\mathcal{A}} \mathtt{a}\ (\mathtt{b}\ x)\ x : q_0$.

▶ Remark. The difference from the Kobayashi-Ong type system [19, 20] is condensed into the definitions of $\mathbb{E}$, $e_{\mathbf{id}}$, *Eff* and $\circ$. Indeed, a variant of the Kobayashi-Ong type system for a parity automaton $(\Sigma, Q, \delta, q_0, \Omega)$ is produced by the following definitions: $\mathbb{E} = \{0, 1, \ldots, M\}$ where $M = \max\{\Omega(q) \mid q \in Q\}$, $e_{\mathbf{id}} = 0$, $\mathit{Eff}(q) = \Omega(q)$, and $e \circ e' = \max\{e, e'\}$. This variant actually deviates from Kobayashi and Ong's type system [20] in the way the priorities of visited states are counted in the rules (Var) and (Const). The variant is an instance of

Tsukada and Ong's type system [38], and is also close to the type system of Grellois and Melliès [10, 9].

The typability of a HORS is defined using *Streett games*.

▶ **Definition 7** (Streett game). A *Streett game* is a quadruple $G = (V_\exists, V_\forall, E, \mathcal{C})$ where $V_\exists$ and $V_\forall$ are disjoint sets of vertices (we write $V = V_\exists \cup V_\forall$), $E \subseteq V \times V$ is a set of edges and $\mathcal{C}$ is a Streett acceptance condition on vertices. A *play* on G from $v_0 \in V_\exists$ is a finite or infinite path from v_0 of the directed graph (V, E). A play is *maximal* if it is infinite, or if it is a finite play $v_0 \dots v_n$ and there is no $v_{n+1} \in V$ such that $(v_n, v_{n+1}) \in E$. A (maximal) play is *winning* either if it is finite and the last vertex of it is in $V_\forall$, or if it is infinite and it satisfies the acceptance condition $\mathcal{C}$, i.e., for each $(E_i, F_i) \in \mathcal{C}$, a vertex in F_i occurs infinitely often whenever a vertex in E_i occurs infinitely often. A *strategy* $\mathcal{W}$ is a partial map from $V^*V_\exists$ to V that is edge-respecting, i.e., for every $\widetilde{v} = v_0 \dots v_n \in dom(\mathcal{W})$, $(v_n, \mathcal{W}(\widetilde{v})) \in E$. A play $\widetilde{v} = v_0 v_1 \dots$ *follows* a strategy $\mathcal{W}$ if for every i, $v_i \in V_\exists$ and $|\widetilde{v}| > i$ imply $\mathcal{W}(v_0 \dots v_i) = v_{i+1}$. A strategy $\mathcal{W}$ is a *winning strategy* from $v_0 \in V_\exists$ if every play that follows $\mathcal{W}$ does not get stuck (i.e., if $\widetilde{v} \in V^*V_\exists$ follows $\mathcal{W}$, then $\widetilde{v} \in dom(\mathcal{W})$), and every maximal play from v_0 that follows $\mathcal{W}$ is winning.

For a HORS $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, we define the *typability game* $G_{\mathcal{G},\mathcal{A}}$ as the Streett game $(V_\exists, V_\forall, E_\exists \cup E_\forall, \mathcal{C}^\uparrow)$ where:

$$
\begin{array}{lcl}
V_\exists & = & \{(F, \theta, e) \mid F \in dom(\mathcal{N}),\ \theta :: \mathcal{N}(F),\ \text{and } e \in \mathbb{E}\} \\
V_\forall & = & \{\Gamma \mid {}^\forall(F : (\theta, e)) \in \Gamma.\ F \in dom(\mathcal{N}),\ \theta :: \mathcal{N}(F),\ \text{and } e \in \mathbb{E}\} \\
E_\exists & = & \{((F, \theta, e), \Gamma) \in V_\exists \times V_\forall \mid \Gamma \vdash_{\mathcal{A}} \mathcal{R}(F) : \theta\} \\
E_\forall & = & \{(\Gamma, (F, \theta, e)) \in V_\forall \times V_\exists \mid (F : (\theta, e)) \in \Gamma\} \\
\mathcal{C}^\uparrow & = & \{(E_1^\uparrow, F_1^\uparrow), \dots, (E_k^\uparrow, F_k^\uparrow)\} \text{ where } E^\uparrow = \{(F, \theta, e) \in V_\exists \mid E \in e\}
\end{array}
$$

Intuitively, in a position (F, θ, e), Player tries to show why F has type θ by giving a type environment Γ such that $\Gamma \vdash_{\mathcal{A}} \mathcal{R}(F) : \theta$; in a position Γ, Opponent tries to challenge Player by picking a type binding from Γ, and asking why that assumption is valid. A HORS $\mathcal{G}$ is *well-typed*, denoted by $\vdash_{\mathcal{A}} \mathcal{G}$, when $G_{\mathcal{G},\mathcal{A}}$ has a winning strategy from $(S, q_0, e_{\mathbf{id}})$.

▶ **Theorem 8** (Correctness). *Given a HORS $\mathcal{G}$ and a Streett automaton $\mathcal{A}$, the value tree of $\mathcal{G}$ is accepted by $\mathcal{A}$ if and only if $\vdash_{\mathcal{A}} \mathcal{G}$.*

We sketch a proof below;[4] See the longer version [36] for more details.

Proof. We are to define a *winning condition* that instantiates Tsukada and Ong's type system for model checking Böhm trees [38] so that the resulting type system is equivalent to our type system and the correctness of our type system follows from the correctness of Tsukada and Ong's type system. A winning condition is a structure $(\mathbb{E}, \mathbb{F}, \Omega)$ where $\mathbb{E}$ and $\mathbb{F}$ are partially ordered sets (we denote both the orders by $\preceq$) and Ω is a downward-closed subset of $\mathbb{F}$, equipped with four operations $\circ : \mathbb{E} \times \mathbb{E} \to \mathbb{E}$, $\circledast : \mathbb{E} \times \mathbb{F} \to \mathbb{F}$, $\pi : \mathbb{E}^\omega \to \mathbb{F}$, and $\setminus : \mathbb{E} \times \mathbb{E} \to \mathbb{E}$ that satisfy additional requirements.

Let $\mathcal{E}$ be defined by $\mathcal{E} = 2^{\{E_1, \dots, E_k, F_1, \dots, F_k\}}$. Let $\mathcal{S}_i : \mathcal{E} \to \{-1, 0, 1\}$ for each $i \in \{1, \dots, k\}$ be defined by: (i) $\mathcal{S}_i(e) = -1$ if $F_i \in e$, (ii) $\mathcal{S}_i(e) = 0$ if $E_i \notin e$ and $F_i \notin e$, and (iii) $\mathcal{S}_i(e) = 1$ otherwise. A preorder $\preceq$ on $\mathcal{E}$ is defined by $e \preceq e' \overset{\text{def}}{\Leftrightarrow} \mathcal{S}_i(e) \leq \mathcal{S}_i(e')$ for every i. It induces an equivalence relation $\sim$ on $\mathcal{E}$ and a partial order $\preceq$ on $\mathcal{E}/\!\sim$. We define a

[4] Here we assume some familiarity with Tsukada and Ong's type system [38].

winning condition $(\mathbb{E}, \mathbb{F}, \Omega)$ that represents a Streett condition $\mathcal{C} = \{(E_1, F_1), \ldots, (E_k, F_k)\}$ as follows: $\mathbb{E} = \mathcal{E}/\sim$, $\mathbb{F} = \{\mathbf{e}, \mathbf{o}\}$ with $\mathbf{e} \preceq \mathbf{o}$, and $\Omega = \{\mathbf{e}\}$. The required operations are defined by $e \circ e' = e \cup e'$, $e \circledast f = f$, $\pi(\widetilde{e}) = \mathbf{e}$ if and only if $\widetilde{e}$ satisfies the Streett acceptance condition, and $e \backslash e' = \bigvee\{d' \mid e \circ d' \preceq e'\}$ where $\bigvee\{d_1, \ldots, d_n\} = d_1 \vee \cdots \vee d_n$ with $e \vee e' = ((nf(e))^{(E)} \cup (nf(e'))^{(E)}) \cup (e^{(F)} \cap e'^{(F)})$. Here, $nf(e)$ is a minimal set such that $e \sim nf(e)$, $e^{(E)} = e \cap \{E_1, \ldots, E_k\}$, and $e^{(F)} = e \cap \{F_1, \ldots, F_k\}$.

It is known that for a HORS $\mathcal{G}$, there is a $\lambda\mathbf{Y}$-calculus term $T_{\mathcal{G}}$ whose Böhm tree is identical to the value tree $[\![\mathcal{G}]\!]$ of $\mathcal{G}$ [33]. By comparing the two type systems, it can be checked that $\vdash_{\mathcal{A}} \mathcal{G} \Leftrightarrow \Gamma_{\mathcal{A}} \vdash_{\mathbf{TO}} T_{\mathcal{G}} : q_0$, where $\vdash_{\mathbf{TO}}$ is the type judgement of $\lambda\mathbf{Y}$-term by Tsukada and Ong and $\Gamma_{\mathcal{A}}$ is the set of all type bindings (for terminals) that can be obtained by our (Const) rule. By the transfer theorem (Theorem 18 in [38]) by Tsukada and Ong, this judgement is equivalent to a type-checking game over a Böhm tree (written $\Gamma_{\mathcal{A}} \models \mathrm{BM}(T_{\mathcal{G}}) : q_0$) in that $\Gamma_{\mathcal{A}} \vdash_{\mathbf{TO}} T_{\mathcal{G}} : q_0$ if and only if this game is winning. Moreover, this game is winning if and only if $[\![\mathcal{G}]\!]$ is accepted by $\mathcal{A}$, which concludes the theorem. ◀

By a discussion similar to [19], the number of the edges of the typability game $G_{\mathcal{G},\mathcal{A}}$ is bounded by $|\mathcal{N}| \cdot \mathbf{exp}_N(O(A|\mathbb{E}||Q|))$ for $N \geq 2$ and $|\mathcal{N}| \cdot 2^{O((A|\mathbb{E}||Q|)^2)}$ for $N = 1$,[5] where $N = \max\{ord(\mathcal{N}(F)) \mid F \in dom(\mathcal{N})\}$ and $A = \max\{arity(\mathcal{N}(F)) \mid F \in dom(\mathcal{N})\}$, and $\mathbf{exp}_n$ is defined by $\mathbf{exp}_0(x) = x$ and $\mathbf{exp}_{n+1}(x) = 2^{\mathbf{exp}_n(x)}$. By means of Piterman and Pneuli's algorithm [30] for Streett game solving, the game can be solved in time $O(|\mathcal{N}|^{k+2}\mathbf{exp}_N(O(A|\mathbb{E}||Q|))kk!)$ where $k = |\mathcal{C}|$. Therefore, the typability game can be solved in time N-fold exponential in A and $|Q|$, and $(N+1)$-fold exponential in k, as $|\mathbb{E}| = 4^k$. If we use $\mathbb{E} = 2^{\{E_1,\ldots,E_k,F_1,\ldots,F_k\}}/\sim$ instead of $\mathbb{E} = 2^{\{E_1,\ldots,E_k,F_1,\ldots,F_k\}}$, where $\sim$ is the equivalence relation defined in the proof of Theorem 8, $|\mathbb{E}|$ is reduced to 3^k.

4 Practical Algorithms for Streett and Parity Automata Model Checking

This section proposes HorSatS and HorSatP, new practical algorithms for Streett and parity automata model checking of HORS, respectively.

The type-based characterization in the previous section actually yields a straightforward model checking algorithm, which first constructs the typability game $G_{\mathcal{G},\mathcal{A}}$ and solves it, as discussed at the end of the previous section. It is, however, impractical since the size of the typability game is huge: N-fold exponential for an order-N HORS. This is because the number of intersection types for order-N functions is N-fold exponential.

Following previous algorithms for trivial automata model checking [14, 16, 3] and parity automata model checking [7, 28], our algorithm for Streett automata model checking first computes a set of types relevant for deciding the model checking problem, constructs a subgame (i.e., a subgraph when viewed as a graph) of $G_{\mathcal{G},\mathcal{A}}$ constructed from those types, and then solves it. If the set of relevant types is sufficiently small for typical instances, the algorithm can be expected to terminate quickly, although in the worst case it still suffers from the N-fold exponential time complexity.

The overall structure of the algorithm HorSatS is shown in Figure 2. An *effectless type environment*, denoted by Θ, is a set of type bindings $F : \theta$ where F is a non-terminal and θ is a prime type. The algorithm starts with a certain initial effectless type environment Θ_0

[5] We use the notation $f(x) = g(O(h(x)))$ to mean that $f(x)$ is bounded by $g(h'(x))$ for some $h'(x)$ with $h'(x) = O(h(x))$.

$\Theta := \Theta_0$
while $(\mathcal{F}(\Theta) \neq \Theta)\{$
  $\Theta := \mathcal{F}(\Theta)$
$\}$
return whether *ConstructGame*(Θ) has a winning strategy

Figure 2 The proposed algorithm HorSatS.

(which will be defined below), and then expands it by repeatedly applying $\mathcal{F}$, until it reaches a fixpoint Θ^{fix}. The algorithm then constructs a subgame of $G_{\mathcal{G},\mathcal{A}}$ consisting of only types occurring in Θ^{fix} and solves it. The algorithm HorSatP also has exactly the same structure; we just need to adapt $\mathcal{F}$ and *ConstructGame* for parity games.

We describe the construction of Θ_0 and the expansion function $\mathcal{F}$ below; it has been inspired by Broadbent and Kobayashi's HorSatT algorithm for trivial automata model checking [3]. Let an input HORS be $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ and an input Streett automaton be $\mathcal{A} = (\Sigma, Q, \delta, q_0, \mathcal{C})$. The initial effectless type environment Θ_0 is defined by:

$$\Theta_0 = \{F : \overbrace{\top \to \cdots \to \top}^{m} \to q \mid F \in \mathit{dom}(\mathcal{N}),\ m = \mathit{arity}(\mathcal{N}(F)),\ q \in Q\}.$$

The expansion function $\mathcal{F}$ is defined by:

$$\begin{aligned}\mathcal{F}(\Theta) =& \Theta \cup \{F : \tau_1 \to \cdots \to \tau_m \to q \mid \mathcal{R}(F) = \lambda x_1 \ldots x_m.\ t,\\ & (\tau_1 \to \cdots \to \tau_m \to q) :: \mathcal{N}(F),\\ & \tau_i \subseteq \mathbf{Types}_\Theta(\mathbf{Flow}(x_i)) \text{ for each } i \in \{1, \ldots, m\},\\ & \Gamma \cup \{x_1 : \tau_1, \ldots, x_m : \tau_m\} \vdash_{\mathcal{A}} t : q\\ & \text{ for some } \Gamma \text{ such that } {}^{\forall}(G : (\theta, e)) \in \Gamma.\ (G : \theta) \in \Theta\}.\end{aligned}$$

Here, $\mathbf{Flow}(x)$ is an overapproximation of the (possibly infinite) set of terms to which x may be bound in a reduction sequence from S; it can be obtained by a flow analysis algorithm like 0CFA. For a set U of terms, $\mathbf{Types}_\Theta(U)$ is defined as $\{\theta \mid \Gamma \vdash_{\mathcal{A}} u : \theta, u \in U, {}^{\forall}(F : (\theta, e)) \in \Gamma.\ (F : \theta) \in \Theta\}$. The notation $\{x_1 : \tau_1, \ldots, x_m : \tau_m\}$ represents the type environment $\{x_i : (\theta_{i,j}, e_{i,j}) \mid i \in \{1, \ldots, m\},\ (\theta_{i,j}, e_{i,j}) \in \tau_i\}$.

Finally, *ConstructGame*(Θ) returns a subgame $G_{\mathcal{G},\mathcal{A},\Theta}$ of $G_{\mathcal{G},\mathcal{A}}$, obtained by restricting $E_\exists$ to the following subset:

$$E'_\exists = \{((F, \theta, e), \Gamma) \in V_\exists \times V_\forall \mid \Gamma \vdash_{\mathcal{A}} \mathcal{R}(F) : \theta \text{ and } {}^{\forall}(G : (\theta, e)) \in \Gamma.\ (G : \theta) \in \Theta\}.$$

The algorithm HorSatP is obtained by (i) replacing the type judgment relation used in the expansion function with that of the Kobayashi-Ong type system, and (ii) modifying *ConstructGame*(Θ) to produce a subgame of the typability game for the Kobayashi-Ong type system. See the longer version [36] for more details.

▶ **Example 9** (a sample run of the algorithm)**.** Consider a HORS $\mathcal{G}_2 = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ where $\Sigma = \{\mathtt{a} \mapsto 2,\ \mathtt{b} \mapsto 1,\ \mathtt{c} \mapsto 0\}$, $\mathcal{N} = \{F \mapsto (\circ \to \circ),\ S \mapsto \circ\}$ and $\mathcal{R} = \{F \mapsto (\lambda x.\mathtt{a}\ x\ (F\ (\mathtt{b}\ x))),\ S \mapsto (F\ \mathtt{c})\}$, and a Streett automaton $\mathcal{A}_2 = (\Sigma, Q, \delta, q_\mathtt{a}, \mathcal{C})$ with Q $= \{q_\mathtt{a}, q_\mathtt{b}\}$, $\mathcal{C} = \{(E_1, \emptyset)\}$ where $E_1 = \{q_\mathtt{a}\}$, and δ is defined by $\delta(q, \mathtt{a}) = (1, q_\mathtt{a}) \wedge (2, q_\mathtt{a})$ for

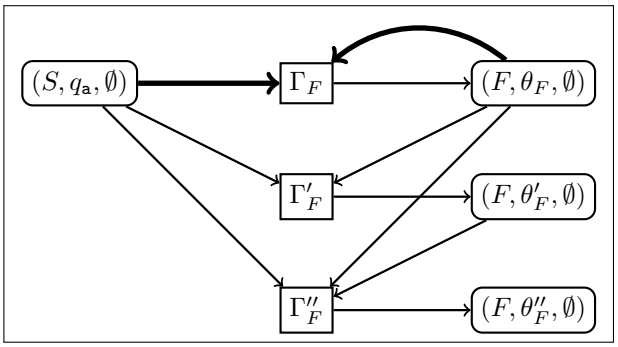


Figure 3 A Streett game generated by HorSatS for inputs $\mathcal{G}_2$ and $\mathcal{A}_2$. Only the part reachable from $(S, q_{\mathtt{a}}, \emptyset)$ is shown. A memoryless winning strategy is indicated by the bold arrows.

each q, $\delta(q, \mathtt{b}) = (1, q_{\mathtt{b}})$ for each q, and $\delta(q, \mathtt{c}) = \mathbf{true}$ for each q. The automaton $\mathcal{A}_2$ accepts trees in which $\mathtt{b}$ occurs only finitely often in every path. The initial effectless type environment is $\Theta_0 = \{F : \top \to q_{\mathtt{a}}, F : \top \to q_{\mathtt{b}}, S : q_{\mathtt{a}}, S : q_{\mathtt{b}}\}$. Let $\mathbf{Flow}(x) = \{\mathtt{b}^n\mathtt{c} \mid n \geq 0\}$ (hence $\mathbf{Types}_\Theta(\mathbf{Flow}(x_i)) = \{q_{\mathtt{a}}, q_{\mathtt{b}}\}$ for any Θ). The fixpoint calculation proceeds as follows.

$$\Theta_1 = \mathcal{F}(\Theta_0) = \Theta_0 \cup \{F : (q_{\mathtt{a}}, \emptyset) \to q_{\mathtt{a}},\ F : (q_{\mathtt{b}}, \{E_1\}) \to q_{\mathtt{b}}\}.$$
$$\Theta_2 = \mathcal{F}(\Theta_1) = \Theta_1 \cup \{F : (q_{\mathtt{a}}, \emptyset) \wedge (q_{\mathtt{b}}, \{E_1\}) \to q_{\mathtt{a}},\ F : (q_{\mathtt{a}}, \emptyset) \wedge (q_{\mathtt{b}}, \{E_1\}) \to q_{\mathtt{b}}\}.$$
$$\Theta_3 = \mathcal{F}(\Theta_2) = \Theta_2.$$

The game constructed by the algorithm is shown in Figure 3, where:

$$\begin{array}{llllll} \theta_F & = (q_{\mathtt{a}}, \emptyset) \wedge (q_{\mathtt{b}}, \{E_1\}) \to q_{\mathtt{a}}. & \theta'_F & = (q_{\mathtt{a}}, \emptyset) \to q_{\mathtt{a}}. & \theta''_F & = \top \to q_{\mathtt{a}}. \\ \Gamma_F & = \{F : (\theta_F, \emptyset)\}. & \Gamma'_F & = \{F : (\theta'_F, \emptyset)\}. & \Gamma''_F & = \{F : (\theta''_F, \emptyset)\}. \end{array}$$

As the game has a winning strategy, the algorithm returns "Yes."

The proposed algorithm is sound and complete, as stated in the following theorems. The soundness (Theorem 10) follows from the fact that *ConstructGame*(Θ) produces a subgame of $G_{\mathcal{G},\mathcal{A}}$ obtained by restricting only Player's moves. The completeness is, however, non-trivial, as to why the fixpoint Θ^{fix} is sufficiently large so that Player can win the subgame $G_{\mathcal{G},\mathcal{A},\Theta^{\mathrm{fix}}}$ if she can win the whole game $G_{\mathcal{G},\mathcal{A}}$. Whilst the construction of Θ_0 and $\mathcal{F}$ is essentially the same as that of HorSatT algorithm, the completeness proof for HorSatS is much more involved than that for HorSatT.

The proofs below apply to both HorSatS and HorSatP. We use the notations $e_{\mathbf{id}}$, *Eff* and $\circ$ for this generalization.

▶ **Theorem 10** (Soundness). *If* HorSatS *(resp.* HorSatP*) returns "Yes" for an input HORS $\mathcal{G}$ and a Streett (resp. parity) automaton $\mathcal{A}$, then $\vdash_{\mathcal{A}} \mathcal{G}$.*

Proof. Suppose that the algorithm returns "Yes." Then, the Streett game $G_{\mathcal{G},\mathcal{A},\Theta^{\mathrm{fix}}}$ has a winning strategy $\mathcal{W}$ from $(S, q_0, e_{\mathbf{id}})$. As $G_{\mathcal{G},\mathcal{A},\Theta^{\mathrm{fix}}}$ is a subgame of $G_{\mathcal{G},\mathcal{A}}$ obtained by only restricting edges in $E_\exists$, $\mathcal{W}$ is also a winning strategy for $G_{\mathcal{G},\mathcal{A}}$. Thus, we have $\vdash_{\mathcal{A}} \mathcal{G}$. ◀

▶ **Theorem 11** (Completeness). *Given an input HORS $\mathcal{G}$ and a Streett (resp. parity) automaton $\mathcal{A}$,* HorSatS *(resp.* HorSatP*) returns "Yes" if $\vdash_{\mathcal{A}} \mathcal{G}$.*

Here we give only a proof sketch. See the longer version [36] for more details.

Proof Sketch. Suppose $\vdash_{\mathcal{A}} \mathcal{G}$. By the definition of $\vdash_{\mathcal{A}} \mathcal{G}$, there exists a finite memory winning strategy $\mathcal{W}$ of the typability game $G_{\mathcal{G},\mathcal{A}}$. By adding a rule for unfolding non-terminals (i.e., a rule for deriving $F : (\theta, e) \vdash_{\mathcal{A}} F : \theta$ from $\Gamma \vdash_{\mathcal{A}} \mathcal{R}(F)$), the typability of HORS can alternatively be described by the existence of a (possibly infinite) type derivation tree Π_0 for $S : q_0$ such that every infinite path in it satisfies the Streett/parity condition derived from that of $\mathcal{A}$.[6] Such a derivation tree Π_0 can be constructed based on $\mathcal{W}$.

We show that we can transform Π_0 to another derivation tree Π_0'' which only uses the types occurring in Θ^{fix} (where Θ^{fix} is the effectless type environment produced by the fixpoint calculation in our algorithm). We first "cut" Π_0 by stopping unfoldings of non-terminals with a certain threshold for the depth of unfoldings, and replace the type of the non-terminal at each "cut" node with $\top \to \cdots \to \top \to q$, treating $\Gamma \vdash F : \underbrace{\top \to \cdots \to \top}_{arity(F)} \to q$ as an "axiom". This axiom corresponds to the initial type environment Θ_0 used in the algorithm. By accordingly reassigning the types in the tree, we have a finite type derivation tree Π_0' that uses only the types in Θ^{fix} computed by the algorithm.

Unfortunately, Π_0' itself does not represent a winning strategy of $G_{\mathcal{G},\mathcal{A},\Theta^{\mathrm{fix}}}$ as it uses the types of non-terminals in Θ_0 as axioms. If we choose a sufficiently large number as the threshold for the depth of unfoldings, however, by matching each node of Π_0' with a corresponding node in Π_0, we can reconstruct a valid (in the sense that every infinite path satisfies the Streett/parity condition) infinite derivation tree Π_0'', by replacing some edges in Π_0' with "back edges". Since Π_0'' has been obtained by only rearranging edges in Π_0', Π_0'' also contains only types in Θ^{fix}. By the correspondence between a valid infinite derivation tree and a winning strategy of the typability game, we obtain a winning strategy $\mathcal{W}'$ for the subgame $G_{\mathcal{G},\mathcal{A},\Theta^{\mathrm{fix}}}$ constructed by $\mathit{Constructgame}(\Theta^{\mathrm{fix}})$. Thus, the algorithm should return "Yes".

Appendix 8 shows an example of the construction of Π_0''. ◀

The algorithm runs in time polynomial in the size of HORS if the other parameters (the largest order and arity of terminals/non-terminals in HORS and the automaton) are fixed. Here, we assume that, as in [37], the linear-time sub-transitive control flow analysis [11] is used for computing the part $\mathbf{Types}_{\Gamma}(\mathbf{Flow}(x_i))$ in $\mathcal{F}$. We also assume that an input HORS is normalized as in [19] so that for each $F \in dom(\mathcal{N})$, $\mathcal{R}(F)$ is of the form $\lambda\widetilde{x}.\, c\,(F_1\,\widetilde{x}_1)\ldots(F_j\,\widetilde{x}_j)$ where $0 \leq j$, $F_1, \ldots, F_j$ are non-terminals, $\widetilde{x}_1, \ldots, \widetilde{x}_j$ are variables and c is a terminal, a non-terminal or a variable. As an increase of the size of HORS caused by this normalization is linear, it does not affect the time complexity result. Upon those assumptions, (i) The size of a type environment is bounded by $O(P)$ where P is the size of an input HORS, and thus the number of the iteration is bounded by $O(P)$. (ii) Calculation of $\mathcal{F}(\Gamma)$ is done in $O(P)$ time. Thus, the fixpoint Θ^{fix} can be calculated in time quadratic in P.[7] Since the size of $\Theta^{\mathrm{fix}}(F)$ is bounded above by a constant (under the fixed-parameter assumption) for each F, the size of (the relevant part of) the typability game is linear in P. Because we have assumed that the automaton is fixed (which also implies that the index k of a Streett automaton or the largest parity of a parity tree automaton is also fixed), the game can also be solved in time polynomial in P. Thus, the whole algorithm runs in polynomial time under the fixed-parameter assumption.

[6] This alternative view of the typability follows easily from the definition of the typability game. Grellois and Melliès [9, 10] have indeed chosen such a formalization for parity tree automata model checking.

[7] Actually, using the technique of [32], Θ^{fix} can be calculated in linear time.

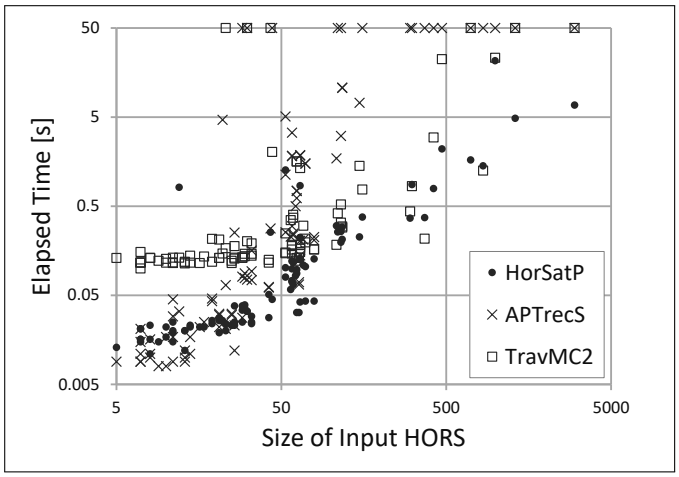


Figure 4 Elapsed time (in seconds) of the model checkers.

5 Experiments

We have implemented HorSatS and HorSatP, Streett and parity automata model checkers for HORS, respectively, based on the algorithm in Section 4. The implementations use a parity game solver PGSolver [6] as a backend for solving the typability game. Although the typability games solved by HorSatS are *Streett* games, they are actually converted to parity games and passed to PGSolver; this is because we could not find a practical implementation of a direct algorithm for Streett game solving.

We have conducted two kinds of experiments, as reported below. The first experiment aims to confirm the effectiveness of the proposed fixed-parameter polynomial time algorithm. To this end, we have compared HorSatP with the previous parity automata model checkers APTRecS [7] and TravMC2 [28]. The second experiment aims to evaluate the effectiveness of the direct approach to Streett automata model checking. To this end, we have compared HorSatS with a combination of HorSatP and a conversion from Streett to parity tree automata.

HorSatP vs APTRecS/TravMC2

We have used a benchmark consisting of 97 inputs of parity automata model checking problems, which include all the inputs used in the evaluation of APTRecS and TravMC2 [7, 28], and also new inputs derived from verification of tree processing programs [39, 22]. The experiment was conducted on a laptop computer with an Intel Core i5-6200U CPU and 8GB of RAM. To achieve the best performance of each model checker, we have used Ubuntu 16.04 LTS for APTRecS and HorSatP, and Windows 10 for TravMC2. Figure 4 shows the results. The horizontal axis is the size (the number of symbols in HORS) of an input, and the vertical axis is the elapsed time of each model checker. The points at the upper edge are timed-out runs (runs that took more than 50 seconds).

For 27 tiny inputs (of size less than 20) APTRecS tends to be the fastest. For the remaining 70 inputs, APTRecS, TravMC2, and HorSatP won 6, 3, and 61 of them respectively. The results indicate that HorSatP usually outperforms the existing model checkers except for tiny inputs. In particular, HorSatP can handle a number of large inputs for which APTRecS and TravMC2 timed-out.

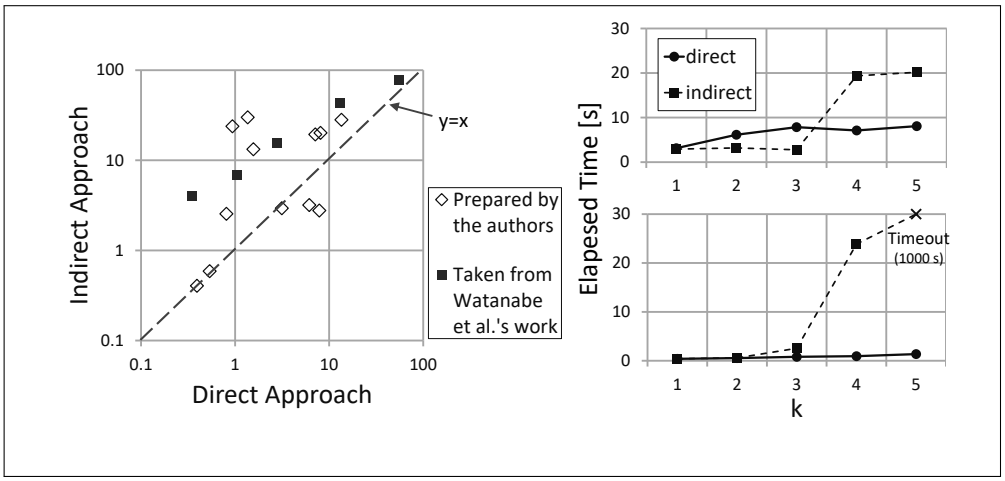


Figure 5 (left) Elapsed time (in seconds) of direct and indirect approaches for several inputs. (right) Elapsed time (in seconds) of direct and indirect approaches for series of inputs with the same structure but different values of k.

Direct vs Indirect Approaches to Streett Automata Model Checking of HORS

We have compared HorSatS with an indirect approach, which first converts an input Streett automaton to a parity automaton by means of the IAR construction [4, 8] and then performs parity automata model checking using HorSatP. The experiment was conducted on a desktop computer with an Intel Core i7-2600 CPU and 8GB of RAM. The OS was Ubuntu 16.04 LTS. We have used two benchmark sets. The first one has been prepared by the authors, hand-made or program-generated. The second one has been taken from Watanabe et al.'s fair non-termination verification tool for functional programs [41], with a slight modification to increase k (the number of pairs in Streett conditions); Watanabe et al.'s original tool supported only the case for $k = 1$. To evaluate the dependency on k, we have tested some of the inputs for different values of k. The results are shown in Figure 5. The left figure shows the elapsed time of both approaches for several inputs. The horizontal axis is that of the direct approach, and the vertical axis is that of the indirect approach. (Thus, plots above the line $y = x$ indicate instances for which the direct approach outperformed the indirect one.) The right figure shows the elapsed time for two series of inputs with the same structure but different values of k. The horizontal axis is the value of k, and the vertical axis is the elapsed time of each approach. The results suggest that the direct approach often outperforms the indirect approach. In particular, the direct approach seems to be noticeably more scalable with respect to k.

6 Related Work

As already mentioned, our type system for Streett automata model checking of HORS presented in Section 3 is a variant of the Kobayashi-Ong type system [19] for parity tree automata model checking, and may also be viewed as an instance of Tsukada and Ong's type system [38], an extension/generalization of the Kobayashi-Ong type system. Our main contribution in this respect is the specific design of "effects" suitable for Streett automata model checking. A naive approach would have been to use a set of *states* (rather than a set consisting of E_i, F_i) as an effect; that would suffer from the $(N + 1)$-fold exponential time

complexity in the size of the automaton, as opposed to N-fold exponential time complexity obtained in the last paragraph of Section 3.

Our algorithms HorSatP and HorSatS are the first practical algorithms for Streett or parity tree automata model checking of HORS which run in time polynomial in the size of HORS (under the fixed-parameter assumption). The previous algorithms APTRecS [7] and TravMC2 [28] for parity tree automata model checking did not satisfy that property. The advantage of the new algorithms has been confirmed also through experiments. For trivial automata model checking, several fixed-parameter polynomial time algorithms have been known, including GTRecS [16], HorSat/HorSatT [3], and Preface [31]. Our algorithms are closest and similar to HorSatT algorithm, although the correctness proof for our new algorithms is much more involved.

Model checking of HORS has been applied to automated verification of higher-order programs [15, 21, 17, 26, 24, 23, 41]. In particular, parity/Streett automata model checking has been applied to liveness verification [24, 7, 41]. Among others, Watanabe et al. [41] reduced the problem of disproving fair termination of functional programs (which is obtained from general liveness verification problems through Vardi's reduction [40]) to Streett automata model checking of HORS, and used an indirect approach to solving the latter by a further reduction to parity automata model checking of HORS. As confirmed through experiments, our direct approach to Streett automata model checking often outperforms their indirect approach.

7 Conclusion

We have proposed a type system and an algorithm for Streett automata model checking of HORS. The main contributions are twofold. First, ours is the first type system and algorithm that can directly be applied to Streett automata model checking of HORS; we have confirmed the advantage of the direct approach through experiments. Secondly, our algorithm HorSatS and its variant HorSatP for parity automata model checking are the first practical algorithms for Streett/parity automata model checking of HORS that run in time polynomial in the size of HORS, under the fixed-parameter assumption. We have also confirmed through experiments that HorSatP often outperforms the previous parity automata model checkers for HORS. Future work includes further optimizations of Streett/parity automata model checkers.

Acknowledgments. We would like to thank anonymous referees for useful comments.

References

1 Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong. The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable. In *TLCA 2005*, volume 3461 of *LNCS*, pages 39–54. Springer, 2005. doi:10.1007/11417170_5.

2 Mohamed Faouzi Atig, Ahmed Bouajjani, Michael Emmi, and Akash Lal. Detecting fair non-termination in multithreaded programs. In *Proc. of CAV 2012*, volume 7358 of *LNCS*, pages 210–226. Springer, 2012. doi:10.1007/978-3-642-31424-7_19.

3 Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In *Proc. of CSL 2013*, volume 23 of *LIPIcs*, pages 129–148, 2013. doi:10.4230/LIPIcs.CSL.2013.129.

4 Nils Buhrke, Helmut Lescow, and Jens Vöge. Strategy construction in infinite ganes with streett and rabin chain winning conditions. In *TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 207–224. Springer, 1996.

5 Byron Cook, Alexey Gotsman, Andreas Podelski, Andrey Rybalchenko, and Moshe Y. Vardi. Proving that programs eventually do something good. In *Proc. of POPL*, pages 265–276. ACM Press, 2007.

6 Oliver Friedmann and Martin Lange. PGSolver. Available at `https://github.com/tcsprojects/pgsolver`.

7 Koichi Fujima, Sohei Ito, and Naoki Kobayashi. Practical alternating parity tree automata model checking of higher-order recursion schemes. In *Proc. of APLAS 2013*, volume 8301 of *LNCS*, pages 17–32, 2013.

8 E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*. LNCS. Springer, 2002.

9 Charles Grellois. *Semantics of linear logic and higher-order model-checking*. PhD thesis, Universit'e Paris Diderot, 2016.

10 Charles Grellois and Paul-André Melliès. Relational semantics of linear logic and higher-order model checking. In *CSL*, volume 41 of *LIPIcs*, pages 260–276. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

11 Nevin Heintze and David McAllester. On the cubic bottleneck in subtyping and flow analysis. In *Proc. of LICS*, pages 342–351, 1997.

12 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In *TLCA*, volume 2044 of *LNCS*, pages 253–267. Springer, 2001.

13 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*, volume 2303 of *LNCS*, pages 205–222. Springer, 2002.

14 Naoki Kobayashi. Model-checking higher-order functions. In *Proc. of PPDP 2009*, pages 25–36. ACM Press, 2009.

15 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proc. of POPL*, pages 416–428. ACM Press, 2009.

16 Naoki Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In *Proc. of FoSSaCS 2011*, volume 6604 of *LNCS*, pages 260–274. Springer, 2011.

17 Naoki Kobayashi. Model checking higher-order programs. *Journal of the ACM*, 60(3), 2013.

18 Naoki Kobayashi. HorSat2: A saturation-based model checker for hors. A tool available from `http://www-kb.is.s.u-tokyo.ac.jp/~koba/horsat2/`, 2016.

19 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proc. of LICS 2009*, pages 179–188. IEEE Computer Society Press, 2009.

20 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. A draft of the longer version of [20], available from `http://www-kb.is.s.u-tokyo.ac.jp/~koba/papers/lics09-full.pdf`, 2012.

21 Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Predicate abstraction and CEGAR for higher-order model checking. In *Proc. of PLDI*, pages 222–233. ACM Press, 2011.

22 Naoki Kobayashi, Naoshi Tabuchi, and Hiroshi Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. In *Proc. of POPL*, pages 495–508. ACM Press, 2010.

23 Takuya Kuwahara, Ryosuke Sato, Hiroshi Unno, and Naoki Kobayashi. Predicate abstraction and CEGAR for disproving termination of higher-order functional programs. In *Proc. of CAV 2015*, volume 9207 of *LNCS*, pages 287–303. Springer, 2015.

24 M. M. Lester, R. P. Neatherway, C.-H. Luke Ong, and S. J. Ramsay. Model checking liveness properties of higher-order functional programs. In *Proc. of ML Workshop*, 2011.

25 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS 2006*, pages 81–90. IEEE Computer Society Press, 2006.

26 C.-H. Luke Ong and Steven Ramsay. Verifying higher-order programs with pattern-matching algebraic data types. In *Proc. of POPL*, pages 587–598. ACM Press, 2011.

27 Akihiro Murase, Tachio Terauchi, Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno. Temporal verification of higher-order functional programs. In *Proc. of POPL 2016*, pages 57–68. ACM, 2016.

28 Robin P. Neatherway and C.-H. Luke Ong. TravMC2: higher-order model checking for alternating parity tree automata. In *Proc. of SPIN 2014*, pages 129–132. ACM, 2014.

29 Robin P. Neatherway, Steven James Ramsay, and C.-H. Luke Ong. A traversal-based algorithm for higher-order model checking. In *Proc. of ICFP 2012*, pages 353–364, 2012.

30 Nir Piterman and Amir Pnueli. Faster solutions of Rabin and Streett games. In *LICS 2006*, pages 275–284. IEEE Computer Society Press, 2006.

31 Steven Ramsay, Robin Neatherway, and C.-H. Luke Ong. An abstraction refinement approach to higher-order model checking. In *Proc. of POPL*, pages 61–72. ACM, 2014.

32 Jakob Rehof and Torben Mogensen. Tractable constraints in finite semilattices. *Science of Computer Programming*, 35(2):191–221, 1999.

33 Sylvain Salvati and Igor Walukiewicz. Recursive schemes, krivine machines, and collapsible pushdown automata. In *Proc. of RP*, volume 7550 of *LNCS*, pages 6–20. Springer, 2012.

34 Ryosuke Sato, Hiroshi Unno, and Naoki Kobayashi. Towards a scalable software model checker for higher-order programs. In *Proc. of PEPM*, pages 53–62. ACM Press, 2013.

35 Robert S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1/2):121–141, 1982.

36 Ryota Suzuki, Koichi Fujima, Naoki Kobayashi, and Takeshi Tsukada. Streett automata model checking of higher-order recursion schemes. A longer version, available from `http://www-kb.is.s.u-tokyo.ac.jp/~koba/papers/fscd17-long.pdf`, 2017.

37 Taku Terao and Naoki Kobayashi. A ZDD-based efficient higher-order model checking algorithm. In *Proc. of APLAS 2014*, volume 8858 of *LNCS*, pages 354–371. Springer, 2014.

38 Takeshi Tsukada and C.-H. Luke Ong. Compositional higher-order model checking via ω-regular games over Böhm trees. In *CSL-LICS 2014*, pages 78:1–78:10. ACM, 2014.

39 Hiroshi Unno, Naoshi Tabuchi, and Naoki Kobayashi. Verification of tree-processing programs via higher-order model checking. In *Proc. of APLAS 2010*, volume 6461 of *LNCS*, pages 312–327. Springer, 2010.

40 Moshe Y. Vardi. Verification of concurrent programs: The automata-theoretic framework. *Ann. Pure Appl. Logic*, 51(1-2):79–98, 1991.

41 Keiichi Watanabe, Ryosuke Sato, Takeshi Tsukada, and Naoki Kobayashi. Automatically disproving fair termination of higher-order functional programs. In *Proc. of ICFP 2016*, pages 243–255. ACM, 2016.

8 Example of the Construction of Π_0'' in the Proof of Theorem 11

We show an example of the construction of the derivation tree Π_0'' explained in the proof sketch of Theorem 11 in Section 4. When showing derivation trees, we omit irrelevant or repeated parts to save space. An application of the new rule for unfolding non-terminals is indicated by (Unfold).

Let $\mathcal{G}_2$ and $\mathcal{A}_2$ be the HORS and the Streett automaton in Example 9 respectively. Let $\mathcal{W}$ be the memoryless winning strategy of $G_{\mathcal{G}_2,\mathcal{A}_2}$ defined by $\mathcal{W}((S, q_{\mathtt{a}}, \emptyset)) = \Gamma_1$ and $\mathcal{W}((F, \hat{\theta}_F, \emptyset)) = \Gamma_1$ where $\hat{\theta}_F = (q_{\mathtt{a}}, \emptyset) \wedge (q_{\mathtt{a}}, \{E_1\}) \wedge (q_{\mathtt{b}}, \{E_1\}) \to q_{\mathtt{a}}$ and $\Gamma_1 = \{F : (\hat{\theta}_F, \emptyset)\}$. We write $F : \theta$ instead of $F : (\theta, \emptyset)$ in type environments. A derivation tree Π_0 that corresponds to $\mathcal{W}$ is as follows:

$$
\dfrac{\dfrac{\emptyset\vdash\mathtt{c}:q_{\mathtt{a}} \quad \emptyset\vdash\mathtt{c}:q_{\mathtt{b}} \quad \dfrac{\dfrac{\dfrac{\dots\vdash\mathtt{a}\;x:(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}} \quad \dfrac{\dfrac{\begin{array}{c}\dfrac{\dfrac{\begin{array}{c}\dfrac{\begin{array}{c}\vdots\\ \{F:\hat\theta_F\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):\hat\theta_F\end{array}}{\{F:\hat\theta_F\}\vdash F:\hat\theta_F}\,(\textsc{Unfold})}{\vdots}\\ \{F:\hat\theta_F\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):\hat\theta_F\end{array}}{\{F:\hat\theta_F\}\vdash F:\hat\theta_F}\,(\textsc{Unfold})}{\vdots}\\ \{F:\hat\theta_F\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):\hat\theta_F\end{array}}{\{F:\hat\theta_F\}\vdash F:\hat\theta_F}\,(\textsc{Unfold}) \quad \dots\vdash\mathtt{b}\;x:q_{\mathtt{a}} \quad \dots\vdash\mathtt{b}\;x:q_{\mathtt{b}}}{\{F:\hat\theta_F, x:(q_{\mathtt{b}},\{E_1\})\}\vdash F\;(\mathtt{b}\;x):q_{\mathtt{a}}}}{\{F:\hat\theta_F, x:q_{\mathtt{a}}, x:(q_{\mathtt{b}},\{E_1\})\}\vdash\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):q_{\mathtt{a}}}}{\{F:\hat\theta_F\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):\hat\theta_F}}{\{F:\hat\theta_F\}\vdash F:\hat\theta_F}\,(\textsc{Unfold})}{\{F:\hat\theta_F\}\vdash F\;\mathtt{c}:q_{\mathtt{a}}}}{\{S:q_{\mathtt{a}}\}\vdash S:q_{\mathtt{a}}}\,(\textsc{Unfold})
$$

We construct Π_0' by "cutting" the derivation tree at a certain threshold of depth, and then reassigning the types in it. Here, we choose to "cut" at the uppermost unfolding shown in the above tree. The resulting tree Π_0' looks like:

$$
\dfrac{\dfrac{\emptyset\vdash\mathtt{c}:q_{\mathtt{a}} \quad \emptyset\vdash\mathtt{c}:q_{\mathtt{b}} \quad \dfrac{\dfrac{\dfrac{\{x:q_{\mathtt{a}}\}\vdash\mathtt{a}\;x:\theta_{\mathtt{a}1} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\{x:(q_{\mathtt{b}},\{E_1\})\}\vdash\mathtt{b}\;x:q_{\mathtt{a}} \quad \dfrac{\dfrac{\dfrac{\{x:q_{\mathtt{a}}\}\vdash\mathtt{a}\;x:\theta_{\mathtt{a}1} \quad \dfrac{\dfrac{}{\{F:\top\to q_{\mathtt{a}}\}\vdash F:\top\to q_{\mathtt{a}}}\,(\textsc{Axiom})}{\{F:\top\to q_{\mathtt{a}}\}\vdash F\;(\mathtt{b}\;x):q_{\mathtt{a}}}}{\{F:\top\to q_{\mathtt{a}}, x:q_{\mathtt{a}}\}\vdash\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):q_{\mathtt{a}}}}{\{F:\top\to q_{\mathtt{a}}\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}}}{\{F:(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}\}\vdash F:(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}}\,(\textsc{Unfold})}{\{F:(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}, x:(q_{\mathtt{b}},\{E_1\})\}\vdash F\;(\mathtt{b}\;x):q_{\mathtt{a}}} \quad \{x:q_{\mathtt{a}}\}\vdash\mathtt{a}\;x:\theta_{\mathtt{a}1}}{\{F:\theta_F, x:q_{\mathtt{a}}, x:(q_{\mathtt{b}},\{E_1\})\}\vdash\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):q_{\mathtt{a}}}}{\{F:\theta_F\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):\theta_F}}{\{F:\theta_F\}\vdash F:\theta_F}\,(\textsc{Unfold}) \quad \{x:(q_{\mathtt{b}},\{E_1\})\}\vdash\mathtt{b}\;x:q_{\mathtt{a}}}{\{F:\theta_F, x:(q_{\mathtt{b}},\{E_1\})\}\vdash F\;(\mathtt{b}\;x):q_{\mathtt{a}}}}{\{F:\theta_F, x:q_{\mathtt{a}}, x:(q_{\mathtt{b}},\{E_1\})\}\vdash\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):q_{\mathtt{a}}}}{\{F:\theta_F\}\vdash\lambda x.\mathtt{a}\;x\;(F\;(\mathtt{b}\;x)):\theta_F}}{\{F:\theta_F\}\vdash F:\theta_F}\,(\textsc{Unfold})}{\{F:\theta_F\}\vdash F\;\mathtt{c}:q_{\mathtt{a}}}}{\{S:q_{\mathtt{a}}\}\vdash S:q_{\mathtt{a}}}\,(\textsc{Unfold})
$$

Here, $\theta_F = (q_{\mathtt{a}},\emptyset)\wedge(q_{\mathtt{b}},\{E_1\})\to q_{\mathtt{a}}$ and $\theta_{\mathtt{a}1} = (q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}$.

The uppermost unfolding has been replaced by the axiom. At the next unfolding below it, F is given the type $(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}$. In the body of this F, x should have type $q_{\mathtt{a}}$ because it is used as an argument of $\mathtt{a}$, which has type $(q_{\mathtt{a}},\emptyset)\to(q_{\mathtt{a}},\emptyset)\to q_{\mathtt{a}}$. On the other hand, x in $\mathtt{b}\;x$ need not have type $q_{\mathtt{b}}$, since $\mathtt{b}\;x$ is not typed in the derivation. Thus, the type

$(q_{\mathtt{a}}, \emptyset) \to q_{\mathtt{a}}$ is assigned to F. By repeating this kind of argument for other unfoldings of F, we update the type of the other occurrences of F to θ_F. Note that Π_0' assigns only the types in Θ^{fix} to non-terminals. In fact, the types $\top \to q_{\mathtt{a}}$, $(q_{\mathtt{a}}, \emptyset) \to q_{\mathtt{a}}$, and θ_F assigned to F at the axiom, the first unfolding (counted from the top), and the second unfolding respectively belong to Θ_0, $\mathcal{F}(\Theta_0)$, and $\mathcal{F}^2(\Theta_0)$.

Notice that, in Π_0', there are two unfolding nodes (the second and third unfolding nodes) labeled by the same judgment $\{F : \theta_F\} \vdash F : \theta_F$. (Note that this is not a coincidence; since there are only finitely many different type judgments, if the threshold for the number of unfoldings is sufficiently large, then there always exist such two nodes in each (sufficiently long) path of Π_0'.) By introducing a "back edge" between them, the following derivation tree Π_0'' is obtained.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{F : \theta_F\} \vdash F : \theta_F}\,(\textsc{Unfold}) \quad \{x : (q_{\mathtt{b}}, \{E_1\})\} \vdash \mathtt{b}\ x : q_{\mathtt{a}}}{\{x : q_{\mathtt{a}}\} \vdash \mathtt{a}\ x : (q_{\mathtt{a}}, \emptyset) \to q_{\mathtt{a}} \quad \{F : \theta_F, x : (q_{\mathtt{b}}, \{E_1\})\} \vdash F\ (\mathtt{b}\ x) : q_{\mathtt{a}}}}{\{F : \theta_F, x : q_{\mathtt{a}}, x : (q_{\mathtt{b}}, \{E_1\})\} \vdash \mathtt{a}\ x\ (F\ (\mathtt{b}\ x)) : q_{\mathtt{a}}}}{\{F : \theta_F\} \vdash \lambda x.\mathtt{a}\ x\ (F\ (\mathtt{b}\ x)) : \theta_F}}{\{F : \theta_F\} \vdash F : \theta_F}\,(\textsc{Unfold})}{\{F : \theta_F\} \vdash F\ \mathtt{c} : q_{\mathtt{a}}}}{\{S : q_{\mathtt{a}}\} \vdash S : q_{\mathtt{a}}}\,(\textsc{Unfold})$$

Note that Π_0'' uses only types that occur in Θ^{fix}. Furthermore, this tree is a valid infinite derivation tree; indeed, it corresponds to the winning strategy of the subgame $G_{\mathcal{G}_2, \mathcal{A}_2, \Theta^{\mathrm{fix}}}$ shown in Figure 3.

A Sequent Calculus for a Semi-Associative Law

Noam Zeilberger

University of Birmingham, Birmingham, UK
noam.zeilberger@gmail.com

Abstract

We introduce a sequent calculus with a simple restriction of Lambek's product rules that precisely captures the classical Tamari order, i.e., the partial order on fully-bracketed words (equivalently, binary trees) induced by a semi-associative law (equivalently, tree rotation). We establish a focusing property for this sequent calculus (a strengthening of cut-elimination), which yields the following coherence theorem: every valid entailment in the Tamari order has exactly one focused derivation. One combinatorial application of this coherence theorem is a new proof of the Tutte–Chapoton formula for the number of intervals in the Tamari lattice Y_n. Elsewhere, we have also used the sequent calculus and the coherence theorem to build a surprising bijection between intervals of the Tamari order and a natural fragment of lambda calculus, consisting of the β-normal planar lambda terms with no closed proper subterms.

1998 ACM Subject Classification F.4.1 Mathematical Logic, G.2.1 Combinatorics

Keywords and phrases proof theory, combinatorics, substructural logic, coherence theorem

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.33

1 Introduction

1.1 The Tamari order, Tamari lattices and associahedra

Suppose you are given a pair of binary trees A and B and the following problem: *transform A into B using only right rotations.* Recall that a right rotation is an operation acting locally on a pair of internal nodes of a binary tree, rearranging them like so:

Solving this problem amounts to showing that $A \leq B$ in the *Tamari order*. Originally introduced by Dov Tamari in the study of monoids with a partially-defined multiplication operation [25, 26, 6], the Tamari order is the partial ordering on words induced by asking that multiplication obeys a *semi-associative law*[1]

$$(A * B) * C \leq A * (B * C)$$

and is monotonic in each argument:

$$\frac{A \leq A'}{A * B \leq A' * B} \qquad \frac{B \leq B'}{A * B \leq A * B'}$$

[1] Clearly, one has to make an arbitrary choice in orienting associativity from left-to-right or right-to-left, and Tamari's original papers in fact took the opposite convention. The literature is inconsistent about this, but since the two possible orders defined are strictly dual it does not make much difference.

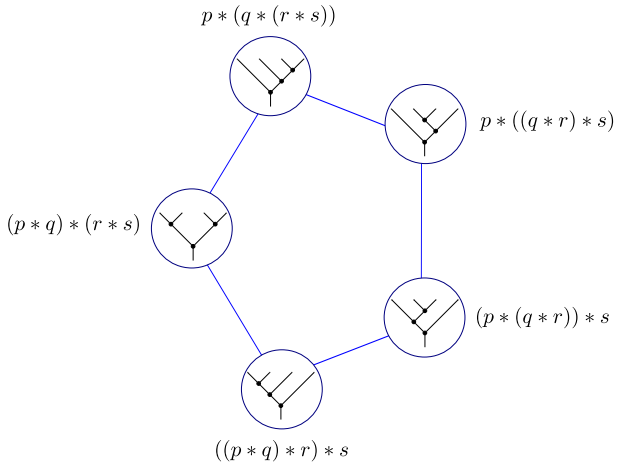
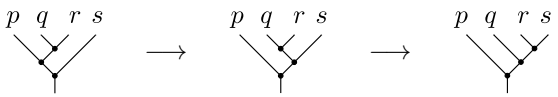


Figure 1 The Tamari lattice Y_3.

For example, the word $(p * (q * r)) * s$ is below the word $p * (q * (r * s))$ in the Tamari order:



The variables $p, q, \ldots$ are just placeholders and what really matters is the underlying shape of such "fully-bracketed words", which is what justifies the above description of the Tamari order in terms of unlabelled binary trees. Since such trees are enumerated by the ubiquitous Catalan numbers (there are $C_n = \binom{2n}{n}/(n+1)$ distinct binary trees with n internal nodes) which also count many other isomorphic families of objects, the Tamari order has many other equivalent formulations as well, such as on strings of balanced parentheses [8], triangulations of a polygon [20], or Dyck paths [2].

For any fixed natural number n, the C_n objects of that size form a lattice under the Tamari order, which is called the *Tamari lattice* Y_n. For example, Figure 1 shows the Hasse diagram of Y_3, which has the shape of a pentagon, and readers familiar with category theory may recognize this as "Mac Lane's pentagon".

More generally, a fascinating property of the Tamari order is that each lattice Y_n generates via its Hasse diagram the underlying graph of an $(n-1)$-dimensional polytope called an "associahedron" [22, 17].

1.2 A Lambekian analysis of the Tamari order

In this paper we will introduce and study a surprisingly elementary presentation of the Tamari order as a *sequent calculus* in the spirit of Lambek [10, 11]. This calculus consists of just one *left rule* and one *right rule:*

$$\frac{A, B, \Delta \longrightarrow C}{A * B, \Delta \longrightarrow C} *L \qquad \frac{\Gamma \longrightarrow A \quad \Delta \longrightarrow B}{\Gamma, \Delta \longrightarrow A * B} *R$$

together with two *structural rules:*

$$\frac{}{A \longrightarrow A}\ id \qquad \frac{\Theta \longrightarrow A \quad \Gamma, A, \Delta \longrightarrow B}{\Gamma, \Theta, \Delta \longrightarrow B}\ cut$$

Here, Γ, Δ, and Θ range over lists of formulas called *contexts*, and we write a comma to indicate concatenation of contexts (which is a strictly associative operation).

In fact, all of these rules come straight from Lambek [10], except for the $*L$ rule which is a restriction of his left rule for products. Lambek's original rule looked like this:

$$\frac{\Gamma, A, B, \Delta \longrightarrow C}{\Gamma, A * B, \Delta \longrightarrow C} *L^{\text{amb}} .$$

That is, Lambek's left rule allowed the formula $A * B$ to appear anywhere in the context, whereas our more restrictive rule $*L$ requires the formula to appear at the leftmost end of the context. It turns out that this simple variation makes all the difference for capturing the Tamari order!

For example, here is a sequent derivation of the entailment $(p*(q*r))*s \leq p*(q*(r*s))$ (we write L and R as short for $*L$ and $*R$, and don't bother labelling instances of id):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p \longrightarrow p} \quad \dfrac{\dfrac{}{q \longrightarrow q} \quad \dfrac{\dfrac{}{r \longrightarrow r} \quad \dfrac{}{s \longrightarrow s}}{r, s \longrightarrow r * s}\, R}{q, r, s \longrightarrow q * (r * s)}\, R}{q * r, s \longrightarrow q * (r * s)}\, L}{p, q * r, s \longrightarrow p * (q * (r * s))}\, R}{p * (q * r), s \longrightarrow p * (q * (r * s))}\, L}{(p * (q * r)) * s \longrightarrow p * (q * (r * s))}\, L \quad .$$

If we had full access to Lambek's original rule then we could also derive the converse entailment (which is false for Tamari):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p \longrightarrow p} \quad \dfrac{\dfrac{}{q \longrightarrow q} \quad \dfrac{}{r \longrightarrow r}}{q, r \longrightarrow q * r}\, R}{p, q, r \longrightarrow p * (q * r)}\, R \quad \dfrac{}{s \longrightarrow s}\, R}{p, q, r, s \longrightarrow (p * (q * r)) * s}}{p, q, r * s \longrightarrow (p * (q * r)) * s}\, L^{\text{amb}}}{p, q * (r * s) \longrightarrow (p * (q * r)) * s}\, L^{\text{amb}}}{p * (q * (r * s)) \longrightarrow (p * (q * r)) * s}\, L \quad .$$

But with the more restrictive rule we can't – the following soundness and completeness result will be established below.

▶ **Claim 1.** *$A \longrightarrow B$ is derivable using the rules $*L$, $*R$, id, and cut if and only if $A \leq B$ holds in the Tamari order.*

As Lambek emphasized, the real power of a sequent calculus comes when it is combined with Gentzen's *cut-elimination* procedure [7]. We will prove the following somewhat stronger form of cut-elimination:

▶ **Claim 2.** *If $\Gamma \longrightarrow A$ is derivable using the rules $*L$, $*R$, id, and cut, then it has a derivation using only $*L$ together with the following restricted forms of $*R$ and id:*

$$\frac{\Gamma^{\text{irr}} \longrightarrow A \quad \Delta \longrightarrow B}{\Gamma^{\text{irr}}, \Delta \longrightarrow A * B} *R^{\text{foc}} \qquad \frac{}{p \longrightarrow p}\, id^{\text{atm}}$$

where Γ^{irr} ranges over contexts that don't have a product $C * D$ at their leftmost end.

This is actually a *focusing completeness* result in the sense of Andreoli [1], and we will refer to derivations constructed using only the rules $*L$, $*R^{\text{foc}}$, and id^{atm} as *focused derivations*. (The above derivation of $(p*(q*r))*s \leq p*(q*(r*s))$ is an example of a focused derivation.) A careful analysis of these three rules confirms that any sequent $\Gamma \longrightarrow A$ has at most one focused derivation.

Combining this property with Claims 1 and 2, we therefore have

▶ **Claim 3.** *Every valid entailment in the Tamari order has exactly one focused derivation.*

This *coherence theorem* is the main contribution of the paper, and we will see that it has several interesting applications.

1.3 The surprising combinatorics of Tamari intervals, planar maps, and planar lambda terms

The original impetus for this work came from wanting to better understand an apparent link between the Tamari order and lambda calculus, which was inferred indirectly via their mutual connection to the combinatorics of embedded graphs.

About a dozen years ago, Fréderic Chapoton [3] proved the following surprising formula for the number of *intervals* in the Tamari lattice Y_n:

$$\frac{2(4n+1)!}{(n+1)!(3n+2)!} \tag{1}$$

Here, by an "interval" of a partially ordered set we just mean a valid entailment $A \leq B$, which can also be identified with the corresponding set of elements $[A, B] = \{\, C \mid A \leq C \leq B \,\}$ (a poset with minimum and maximum elements). For example, the Tamari lattice Y_3 displayed in Figure 1 contains 13 intervals. Chapoton used generating function techniques to show that (1) gives the number of intervals in Y_n, and we will explain how the above coherence theorem can be used to give a new proof of this result. As Chapoton mentions, though, the formula itself did not come out of thin air, but rather was found by querying the *On-Line Encyclopedia of Integer Sequences* (OEIS) [21]. Formula (1) is included in OEIS entry A000260, and in fact it was derived over half a century ago by the graph theorist Bill Tutte [27] for a seemingly unrelated family of objects: it counts the number of (3-connected, rooted) triangulations of the sphere with $3(n+1)$ edges. The same formula is also known to count other natural families of embedded graphs, and in particular it counts the number of bridgeless rooted planar maps with n edges [28].[2] Sparked by Chapoton's observation, Bernardi and Bonichon [2] found an explicit bijection between intervals of the Tamari order and 3-connected rooted planar triangulations, and quite recently, Fang [5] has proposed new bijections between these three different families of objects (i.e., between 3-connected rooted planar triangulations, bridgeless rooted planar maps, and Tamari intervals).

Meanwhile, in [32], Alain Giorgetti and I gave a bijection between rooted planar maps (possibly containing bridges) and a simple fragment of linear lambda calculus consisting of the β-normal planar terms. As with Chapoton's result, this connection between maps and lambda calculus was found using hints from the OEIS, since the sequence enumerating rooted

[2] A *rooted planar map* is a connected graph embedded in the 2-sphere or the plane, with one half-edge chosen as the root. A (rooted planar) *triangulation* (dually, trivalent map) is a (rooted planar) map in which every face (dually, vertex) has degree three. A map is said to be *bridgeless* (respectively, *3-connected*) if it has no edge (respectively, pair of vertices) whose removal disconnects the underlying graph. (Cf. [12].)

planar maps was already known – and once again this sequence was first computed by Tutte, who derived another simple closed formula for the number of rooted planar maps with n edges ($\frac{2(2n)!3^n}{n!(n+2)!}$). Now, let us say that a term is *indecomposable* if it has *no closed proper subterms*. Although this property may be unfamiliar to some readers, indecomposability turns out to be very natural to consider in a linear context – for example, in [30] it was used to give a lambda calculus reformulation of the Four Color Theorem, based on a characterization of bridgeless rooted trivalent maps as indecomposable linear lambda terms. In any case, it is not difficult to check that the bijection described in [32] restricts to a bijection between bridgeless rooted planar maps and indecomposable β-normal planar terms, and therefore that formula (1) also enumerates indecomposable β-normal planar terms by size. It is then a natural question whether there exists a direct bijection between such terms and intervals of the Tamari order.

An explicit bijection between indecomposable β-normal planar terms and Tamari intervals was given in an earlier, longer version of this paper [31], and the proof of its correctness relies in an essential way on the sequent calculus and coherence theorem. That bijection is omitted here in part due to space constraints, and in part because I would like to give a fuller and more conceptual account of the combinatorics, connecting it to the duality between *skew-monoidal categories* [24] and *skew-closed categories* [23]. However, even without this original application, I believe that the sequent calculus considered here is of intrinsic mathematical interest: Tamari lattices and related associahedra have been studied for over sixty years, so the fact that such an elementary and natural proof-theoretic characterization of semi-associativity has been seemingly overlooked is surprising. Moreover, this characterization is clearly productive, since it leads to a much more structured proof of Chapoton's seminal result – an unexpected link between important ideas in proof theory (such as cut-elimination and focusing) and an active research topic in combinatorics.

2 A sequent calculus for the Tamari order

2.1 Definitions and terminology

For reference, we recall here the definition of the sequent calculus introduced in 1.2, and clarify some notational conventions. The four rules of the sequent calculus are:

$$\frac{A, B, \Delta \longrightarrow C}{A * B, \Delta \longrightarrow C} *L \qquad \frac{\Gamma \longrightarrow A \quad \Delta \longrightarrow B}{\Gamma, \Delta \longrightarrow A * B} *R$$

$$\frac{}{A \longrightarrow A}\ id \qquad \frac{\Theta \longrightarrow A \quad \Gamma, A, \Delta \longrightarrow B}{\Gamma, \Theta, \Delta \longrightarrow B}\ cut$$

Uppercase Latin letters ($A, B, \dots$) range over **formulas**, which can be either **compound** ($A * B$) or **atomic** (ranged over by lowercase Latin letters $p, q, \dots$). Uppercase Greek letters $\Gamma, \Delta, \dots$ range over **contexts**, which are (possibly empty) lists of formulas, with concatenation of contexts indicated by a comma. (Let us emphasize that as in Lambek's system [10] but in contrast to Gentzen's original sequent calculus [7], there are no rules of "weakening", "contraction", or "exchange", so the order and the number of occurrences of a formula within a context matters.) A **sequent** is a pair of a context Γ and a formula A.

Abstractly, a **derivation** is a tree (technically, a *rooted planar tree with boundary*) whose internal nodes are labelled by the names of rules and whose edges are labelled by sequents satisfying the constraints indicated by the given rule. The **conclusion** of a derivation is the sequent labelling its outgoing root edge, while its **premises** are the sequents labelling any incoming leaf edges. A derivation with no premises is said to be **closed**.

We write "$\Gamma \longrightarrow A$" as a notation for sequents, but also sometimes as a shorthand to indicate that the given sequent is **derivable** using the above rules, in other words that there exists a closed derivation whose conclusion is that sequent (it will always be clear which of these two senses we mean). Sometimes we will need to give an explicit name to a derivation with a given conclusion, in which case we place it over the sequent arrow.

As in 1.2, when constructing derivations we sometimes write L and R as shorthand for $*L$ and $*R$, and usually don't bother labelling the instances of *id* and *cut* since they are clear from context.

Finally, define the **frontier** $\mathrm{fr}(A)$ of a formula A to be the ordered list of atoms occurring in A (i.e., by $\mathrm{fr}(A*B) = \mathrm{fr}(A), \mathrm{fr}(B)$ and $\mathrm{fr}(p) = p$), and the frontier of a context $\Gamma = A_1, \ldots, A_n$ as the concatenation of frontiers $\mathrm{fr}(\Gamma) = \mathrm{fr}(A_1), \ldots, \mathrm{fr}(A_n)$. The following properties are immediate by examination of the four sequent calculus rules.

▶ **Proposition 4.** *Suppose that* $\Gamma \longrightarrow A$. *Then*

1. (Refinement) $\mathrm{fr}(\Gamma) = \mathrm{fr}(A)$.
2. (Relabelling) $\sigma\Gamma \longrightarrow \sigma A$, *where* σ *is any function sending atoms to atoms.*

2.2 Completeness

We begin by establishing completeness relative to the Tamari order, which is the easier direction.

▶ **Theorem 5** (Completeness)**.** *If* $A \leq B$ *then* $A \longrightarrow B$.

Proof. We must show that the relation $A \longrightarrow B$ is reflexive and transitive, and that the multiplication operation satisfies a semi-associative law and is monotonic in each argument. All of these properties are straightforward:

1. Reflexivity: immediate by *id*.
2. Transitivity: immediate by *cut*.
3. Semi-associativity:

$$\dfrac{\dfrac{\dfrac{}{A \longrightarrow A} \quad \dfrac{\overline{B \longrightarrow B} \quad \overline{C \longrightarrow C}}{B, C \longrightarrow B * C}\, R}{A, B, C \longrightarrow A * (B * C)}\, R}{\dfrac{A * B, C \longrightarrow A * (B * C)}{(A * B) * C \longrightarrow A * (B * C)}\, L}\, L$$

4. Monotonicity:

$$\dfrac{A \longrightarrow A' \quad \overline{B \longrightarrow B}}{\dfrac{A, B \longrightarrow A' * B}{A * B \longrightarrow A' * B}\, L}\, R \qquad \dfrac{\overline{A \longrightarrow A} \quad B \longrightarrow B'}{\dfrac{A, B \longrightarrow A * B'}{A * B \longrightarrow A * B'}\, L}\, R$$

◀

2.3 Soundness

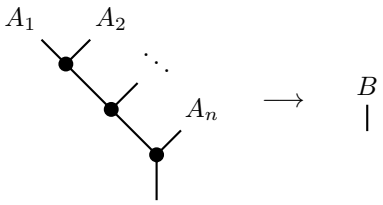
To prove soundness relative to the Tamari order, first we have to explain the interpretation of general sequents. The basic idea is that we can interpret a *non-empty context* as a *left-associated product*. Thus, a general sequent of the form

$$A_1, A_2, \ldots, A_n \longrightarrow B$$

(where $n \geq 1$) is interpreted as an entailment of the form

$$(\cdots(A_1 * A_2) * \cdots) * A_n \leq B$$

in the Tamari order. Visualizing everything in terms of binary trees, the sequent can be interpreted like so:



That is, the context provides information about the *left-branching spine* of the tree which is below in the Tamari order.

Let $\phi[-]$ be the operation taking any non-empty context Γ to a formula $\phi[\Gamma]$ by the above interpretation. The operation is defined by the following equations:

$$\phi[A] = A \qquad \phi[\Gamma, A] = \phi[\Gamma] * A$$

Critical to soundness of the sequent calculus is the following "colax" property of $\phi[-]$:

▶ **Proposition 6.** *$\phi[\Gamma, \Delta] \leq \phi[\Gamma] * \phi[\Delta]$ for all non-empty contexts Γ and Δ.*

Proof. By induction on Δ. The case of a singleton context $\Delta = A$ is immediate. Otherwise, if $\Delta = (\Delta', A)$, we have

$$\phi[\Gamma, \Delta', A] = \phi[\Gamma, \Delta'] * A \leq (\phi[\Gamma] * \phi[\Delta']) * A \leq \phi[\Gamma] * (\phi[\Delta'] * A) = \phi[\Gamma] * \phi[\Delta', A]$$

where the first inequality is by the inductive hypothesis and monotonicity, while the second inequality is by the semi-associative law. ◀

The operation $\phi[-]$ can also be equivalently described in terms of a *right action* $A \circledast \Delta$ of an arbitrary context on a formula, where this action is defined by the following equations:

$$A \circledast \cdot = A \qquad A \circledast (\Delta, B) = (A \circledast \Delta) * B$$

We will make use of a few simple properties of $- \circledast \Delta$:

▶ **Proposition 7.** *$\phi[\Gamma, \Delta] = \phi[\Gamma] \circledast \Delta$ for all non-empty contexts Γ and arbitrary contexts Δ.*

▶ **Proposition 8** (Monotonicity)**.** *If $A \leq A'$ then $A \circledast \Delta \leq A' \circledast \Delta$.*

Proof. Both properties are immediate by induction on Δ, where in the case of Prop. 8 we apply monotonicity of the operations $- * B$. ◀

We are now ready to prove soundness.

▶ **Theorem 9** (Soundness)**.** *If $\Gamma \longrightarrow A$ then $\phi[\Gamma] \leq A$.*

Proof. By induction on the (closed) derivation of $\Gamma \longrightarrow A$. There are four cases, corresponding to the four rules of the sequent calculus:

- (Case $*L$): The derivation ends in

$$\frac{A, B, \Delta \longrightarrow C}{A * B, \Delta \longrightarrow C} \; *L$$

  By induction we have $\phi[A, B, \Delta] \leq C$, but by Prop. 7 we have $\phi[A*B, \Delta] = \phi[A * B] \circledast \Delta = (A * B) \circledast \Delta = \phi[A, B] \circledast \Delta = \phi[A, B, \Delta]$.
- (Case $*R$): The derivation ends in

$$\frac{\Gamma \longrightarrow A \quad \Delta \longrightarrow B}{\Gamma, \Delta \longrightarrow A * B} \; *R$$

  By induction we have $\phi[\Gamma] \leq A$ and $\phi[\Delta] \leq B$, hence $\phi[\Gamma, \Delta] \leq \phi[\Gamma] * \phi[\Delta] \leq A * B$ where we apply Prop. 6 for the first inequality, and monotonicity for the second.
- (Case id): Immediate by reflexivity.
- (Case cut): The derivation ends in

$$\frac{\Theta \longrightarrow A \quad \Gamma, A, \Delta \longrightarrow B}{\Gamma, \Theta, \Delta \longrightarrow B} \; cut$$

  Then $\phi[\Gamma, \Theta, \Delta] = \phi[\Gamma, \Theta] \circledast \Delta \leq (\phi[\Gamma] * \phi[\Theta]) \circledast \Delta \leq (\phi[\Gamma] * A) \circledast \Delta = \phi[\Gamma, A, \Delta] \leq B$. ◀

2.4 Focusing completeness

Cut-elimination theorems are a staple of proof theory, and often provide a rich source of information about a given logic. In this section we will prove a *focusing completeness* theorem, which is an even stronger form of cut-elimination originally formulated by Andreoli in the setting of linear logic [1].

▶ **Definition 10.** A context Γ is said to be **reducible** if its leftmost formula is compound, and **irreducible** otherwise. A sequent $\Gamma \longrightarrow A$ is said to be:
- **left-inverting** if Γ is reducible;
- **right-focusing** if Γ is irreducible and A is compound;
- **atomic** if Γ is irreducible and A is atomic.

▶ **Proposition 11.** *Any sequent is either left-inverting, right-focusing, or atomic.*

▶ **Definition 12.** A closed derivation $\mathcal{D}$ is said to be **focused** if left-inverting sequents only appear as the conclusions of $*L$, right-focusing sequents only as the conclusions of $*R$, and atomic sequents only as the conclusions of id.

We write "Γ^{irr}" to indicate that a context Γ is irreducible.

▶ **Proposition 13.** *A closed derivation is focused if and only if it is constructed using only $*L$ and the following restricted forms of $*R$ and id (and no instances of cut):*

$$\frac{\Gamma^{\text{irr}} \longrightarrow A \quad \Delta \longrightarrow B}{\Gamma^{\text{irr}}, \Delta \longrightarrow A * B} \; *R^{\text{foc}} \qquad \frac{}{p \longrightarrow p} \; id^{\text{atm}}$$

▶ **Example 14.** One way to derive $((p * q) * r) * s \longrightarrow p * ((q * r) * s)$ is by cutting together the two derivations

$$\frac{\dfrac{\mathcal{SA}_{p,q,r}}{(p * q) * r \longrightarrow p * (q * r)} \quad \dfrac{}{s \longrightarrow s}}{\dfrac{(p * q) * r, s \longrightarrow (p * (q * r)) * s}{((p * q) * r) * s \longrightarrow (p * (q * r)) * s}\; L}\; R \qquad \text{and} \qquad \frac{\mathcal{SA}_{p,q*r,s}}{(p * (q * r)) * s \longrightarrow p * ((q * r) * s)}$$

where $\mathcal{SA}_{A,B,C}$ is the derivation of the semi-associative law $(A * B) * C \longrightarrow A * (B * C)$ from the proof of Theorem 5. Clearly this is *not* a focused derivation (besides the *cut* rule, it also uses instances of $*R$ and *id* with a left-inverting conclusion). However, it is possible to give a focused derivation of the same sequent:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p \longrightarrow p} \quad \dfrac{\dfrac{\dfrac{}{q \longrightarrow q} \quad \dfrac{}{r \longrightarrow r}}{q, r \longrightarrow q * r}\, R \quad \dfrac{}{s \longrightarrow s}}{q, r, s \longrightarrow (q * r) * s}\, R}{p, q, r, s \longrightarrow p * ((q * r) * s)}\, R}{p * q, r, s \longrightarrow p * ((q * r) * s)}\, L}{(p * q) * r, s \longrightarrow p * ((q * r) * s)}\, L}{((p * q) * r) * s \longrightarrow p * ((q * r) * s)}\, L$$

In the below, we write "$\Gamma \Longrightarrow A$" as a shorthand notation to indicate that $\Gamma \longrightarrow A$ has a (closed) focused derivation, and "$\mathcal{D} : A \Longrightarrow B$" to indicate that $\mathcal{D}$ is a particular focused derivation of $A \longrightarrow B$.

▶ **Theorem 15** (Focusing completeness). *If* $\Gamma \longrightarrow A$ *then* $\Gamma \Longrightarrow A$.

To prove the focusing completeness theorem, it suffices to show that the *cut* rule as well as the unrestricted forms of *id* and $*R$ are all *admissible* for focused derivations, in the proof-theoretic sense that given focused derivations of their premises, we can obtain a focused derivation of their conclusion. We begin by proving a *focused deduction* lemma (cf. [19]), which entails the admissibility of *id*, then show *cut* and $*R$ in turn.

▶ **Lemma 16** (Deduction). *If* $\Gamma^{\text{irr}} \Longrightarrow A$ *implies* $\Gamma^{\text{irr}}, \Delta \Longrightarrow B$ *for all* Γ^{irr}, *then* $A, \Delta \Longrightarrow B$. *In particular,* $A \Longrightarrow A$.

Proof. By induction on the formula A:

- (Case $A = p$): Immediate by assumption, taking $\Gamma^{\text{irr}} = p$ and $p \Longrightarrow p$ derived by id^{atm}.
- (Case $A = A_1 * A_2$): By composing with the $*L$ rule,

$$\dfrac{A_1, A_1, \Delta \longrightarrow B}{A_1 * A_2, \Delta \longrightarrow B}\, *L$$

we reduce the problem to showing $A_1, A_2, \Delta \Longrightarrow B$, and by the i.h. on A_1 it suffices to show that $\Gamma_1^{\text{irr}} \Longrightarrow A_1$ implies $\Gamma_1^{\text{irr}}, A_2, \Delta \Longrightarrow B$ for all contexts Γ_1^{irr}. Let $\mathcal{D}_1 : \Gamma_1^{\text{irr}} \Longrightarrow A_1$. We can derive $\Gamma_1^{\text{irr}}, A_2 \Longrightarrow A_1 * A_2$ by

$$\mathcal{D} \quad = \quad \dfrac{\overset{\mathcal{D}_1}{\Gamma_1^{\text{irr}} \longrightarrow A_1} \quad \overset{\mathcal{D}_2}{A_2 \longrightarrow A_2}}{\Gamma_1^{\text{irr}}, A_2 \longrightarrow A_1 * A_2}\, *R$$

where we apply the i.h. on A_2 to obtain $\mathcal{D}_2$. Finally, applying the assumption to $\mathcal{D}$ (with $\Gamma^{\text{irr}} = \Gamma_1^{\text{irr}}, A_2$) we obtain the desired derivation of $\Gamma_1^{\text{irr}}, A_2, \Delta \Longrightarrow B$. ◀

▶ **Lemma 17** (Cut). *If* $\Theta \Longrightarrow A$ *and* $\Gamma, A, \Delta \Longrightarrow B$ *then* $\Gamma, \Theta, \Delta \Longrightarrow B$.

Proof. Let $\mathcal{D} : \Theta \Longrightarrow A$ and $\mathcal{E} : \Gamma, A, \Delta \Longrightarrow B$. We proceed by a lexicographic induction, first on the cut formula A and then on the pair of derivations $(\mathcal{D}, \mathcal{E})$ (i.e., at each inductive step of the proof, either A gets smaller, or it stays the same as one of $\mathcal{D}$ or $\mathcal{E}$ gets smaller while the other stays the same).

In the case that $A = p$ we can apply the "frontier refinement" property (Prop. 4) to deduce that $\Theta = p$, so the cut is trivial and we just reuse the derivation $\mathcal{E} : \Gamma, p, \Delta \Longrightarrow B$. Otherwise we have $A = A_1 * A_2$ for some A_1, A_2, and we proceed by case-analyzing the root rule of $\mathcal{E}$:

- (Case id^{atm}): Impossible since A is non-atomic.
- (Case $*R^{\text{foc}}$): This case splits in two possibilities:

  1. $\exists \Delta_1, \Delta_2$ such that $\Delta = \Delta_1, \Delta_2$ and

$$\mathcal{E} \quad = \quad \dfrac{\overset{\mathcal{E}_1}{\Gamma^{\text{irr}}, A, \Delta_1 \longrightarrow B_1} \quad \overset{\mathcal{E}_2}{\Delta_2 \longrightarrow B_2}}{\Gamma^{\text{irr}}, A, \Delta_1, \Delta_2 \longrightarrow B_1 * B_2} *R$$

  2. $\exists \Gamma_1^{\text{irr}}, \Gamma_2$ such that $\Gamma^{\text{irr}} = \Gamma_1^{\text{irr}}, \Gamma_2$ and

$$\mathcal{E} \quad = \quad \dfrac{\overset{\mathcal{E}_1}{\Gamma_1^{\text{irr}} \longrightarrow B_1} \quad \overset{\mathcal{E}_2}{\Gamma_2, A, \Delta \longrightarrow B_2}}{\Gamma_1^{\text{irr}}, \Gamma_2, A, \Delta \longrightarrow B_1 * B_2} *R$$

  In the first case, we cut $\mathcal{D}$ with $\mathcal{E}_1$ to obtain $\Gamma^{\text{irr}}, \Theta, \Delta_1 \Longrightarrow B_1$, then recombine that with $\mathcal{E}_2$ using $*R^{\text{foc}}$ to obtain $\Gamma^{\text{irr}}, \Theta, \Delta_1, \Delta_2 \Longrightarrow B_1 * B_2$. The second case is similar.
- (Case $*L$): This case splits into two possibilities:

  1. $\exists C_1, C_2, \Gamma'$ such that $\Gamma = C_1 * C_2, \Gamma'$ and

$$\mathcal{E} \quad = \quad \dfrac{\overset{\mathcal{E}'}{C_1, C_2, \Gamma', A, \Delta \longrightarrow B}}{C_1 * C_2, \Gamma', A, \Delta \longrightarrow B} *L$$

  We cut $\mathcal{D}$ into $\mathcal{E}'$ and reapply the $*L$ rule.

  2. $\Gamma = \cdot$ and

$$\mathcal{E} \quad = \quad \dfrac{\overset{\mathcal{E}'}{A_1, A_2, \Delta \longrightarrow B}}{A_1 * A_2, \Delta \longrightarrow B} *L$$

  We further analyze the root rule of $\mathcal{D}$:
  - (Case $*L$): $\exists C_1, C_2, \Theta'$ s.t. $\Theta = C_1 * C_2, \Theta'$ and

$$\mathcal{D} \quad = \quad \dfrac{\overset{\mathcal{D}'}{C_1, C_2, \Theta' \longrightarrow A_1 * A_2}}{C_1 * C_2, \Theta' \longrightarrow A_1 * A_2} *L$$

    We cut $\mathcal{D}'$ into $\mathcal{E}$ and reapply the $*L$ rule.
  - (Case id^{atm}): Impossible.
  - (Case $*R^{\text{foc}}$): $\exists \Theta_1^{\text{irr}}, \Theta_2$ s.t. $\Theta^{\text{irr}} = \Theta_1^{\text{irr}}, \Theta_2$ and

$$\mathcal{D} \quad = \quad \dfrac{\overset{\mathcal{D}_1}{\Theta_1^{\text{irr}} \longrightarrow A_1} \quad \overset{\mathcal{D}_2}{\Theta_2 \longrightarrow A_2}}{\Theta_1^{\text{irr}}, \Theta_2 \longrightarrow A_1 * A_2} *R$$

    We cut both $\mathcal{D}_1$ and $\mathcal{D}_2$ into $\mathcal{E}'$ (the cuts are at smaller formulas so the order doesn't matter). ◀

▶ **Lemma 18** ($*R$ admiss.). *If* $\Gamma \Longrightarrow A$ *and* $\Delta \Longrightarrow B$ *then* $\Gamma, \Delta \Longrightarrow A * B$.

Proof. Let $\mathcal{D} : \Gamma \Longrightarrow A$ and $\mathcal{E} : \Delta \Longrightarrow B$. We proceed by induction on $\mathcal{D}$. If Γ is irreducible then we directly apply $*R^{\text{foc}}$. Otherwise, $\mathcal{D}$ must be of the form

$$\mathcal{D} \quad = \quad \dfrac{\begin{array}{c}\mathcal{D}' \\ C_1, C_2, \Gamma' \longrightarrow A\end{array}}{C_1 * C_2, \Gamma' \longrightarrow A} *L$$

so we apply the i.h. to $\mathcal{D}'$ with $\mathcal{E}$, and reapply the $*L$ rule. ◀

Proof of Theorem 15. An arbitrary closed derivation can be turned into a focused one by starting at the top of the derivation tree and using the above lemmas to interpret any instance of the *cut* rule and of the unrestricted forms of *id* and $*R$. ◀

Finally, we mention two simple applications of the focusing completeness theorem.

▶ **Proposition 19** (Frontier invariance). *Let σ be any function sending atoms to atoms. Then $\Gamma \longrightarrow A$ if and only if $\text{fr}(\Gamma) = \text{fr}(A)$ and $\sigma\Gamma \longrightarrow \sigma A$.*

Proof. The forward direction is Prop. 4. For the backward direction we use induction on focused derivations, which is justified by Theorem 15. The only interesting case is $*R^{\text{foc}}$, where we can assume $\text{fr}(\Gamma, \Delta) = \text{fr}(A * B)$ and $\sigma\Gamma \Longrightarrow \sigma A$ ($\sigma\Gamma$ irreducible) and $\sigma\Delta \Longrightarrow \sigma B$. By Prop. 4 we have $\text{fr}(\sigma\Gamma) = \text{fr}(\sigma A)$ and $\text{fr}(\sigma\Delta) = \text{fr}(\sigma B)$, but then elementary properties of lists imply that $\text{fr}(\Gamma) = \text{fr}(A)$ and $\text{fr}(\Delta) = \text{fr}(A)$, from which $\Gamma \Longrightarrow A$ (Γ irreducible) and $\Delta \Longrightarrow B$ follow by the induction hypothesis, hence $\Gamma, \Delta \Longrightarrow A * B$. ◀

If we let $\sigma = \_ \mapsto p$ be any constant relabelling function, then speaking in terms of the Tamari order, Proposition 19 says that to check that two "fully-bracketed words" (a.k.a., formulas) are related, it suffices to check that their frontiers are equal and that the unlabelled binary trees describing their underlying multiplicative structure are related. Although this fact is intuitively obvious, trying to prove it directly by induction on general derivations fails, because in the case of the *cut* rule we cannot assume anything about the frontier of the cut formula A.

▶ **Definition 20.** We say that an irreducible context Γ^{irr} is a **maximal decomposition** of A if $\Gamma^{\text{irr}} \longrightarrow A$, and for any other Θ^{irr}, $\Theta^{\text{irr}} \longrightarrow A$ implies $\Theta^{\text{irr}} \longrightarrow \phi[\Gamma^{\text{irr}}]$.

▶ **Proposition 21.** *If Γ^{irr} is a maximal decomposition of A, then $A, \Delta \longrightarrow B$ if and only if $\Gamma^{\text{irr}}, \Delta \longrightarrow B$.*

Proof. The forward direction is by cutting with $\Gamma^{\text{irr}} \longrightarrow A$, the backwards direction is by the deduction lemma (16) and the universal property of Γ^{irr}. ◀

▶ **Proposition 22.** *Let $\psi[A]$ be the irreducible context defined inductively by:*

$$\psi[p] = p \qquad \psi[A * B] = \psi[A], B$$

Then $\psi[A]$ is a maximal decomposition of A.

Proof. We construct $\psi[A] \longrightarrow A$ by induction on A, and prove the universal property of $\psi[A]$ by induction on focused derivations of $\Theta^{\text{irr}} \longrightarrow A$. ◀

▶ **Proposition 23.** *$\phi[\psi[A]] = A$ and $\psi[\phi[\Theta^{\text{irr}}]] = \Theta^{\text{irr}}$.*

The maximal decomposition $\psi[A]$ of A is essentially the same thing as what Chapoton [3] calls a "décomposition maximale" of a binary tree. The logical characterization expressed in Defn. 20 is quite general, though, and is familiar from studies of focusing in other settings (cf. [29]).

2.5 The coherence theorem

We now come to our main result:

▶ **Theorem 24** (Coherence). *Every derivable sequent has exactly one focused derivation.*

Coherence is a direct consequence of focusing completeness and the following lemma:

▶ **Lemma 25.** *For any context* Γ *and formula* A*, there is at most one focused derivation of* $\Gamma \longrightarrow A$*.*

Proof. We proceed by a well-founded induction on sequents, which can be reduced to multiset induction as follows. Define the **size** $|A|$ of a formula A by $|A * B| = 1 + |A| + |B|$ and $|p| = 0$. (That is, $|A|$ counts the number of multiplication operations occurring in A.) Then any sequent $A_1, \ldots, A_n \longrightarrow B$ induces a multiset of sizes $(\biguplus_{i=1}^{n} |A_i|) \uplus |B|$, and at each step of our induction this multiset will decrease in the multiset ordering.

There are three cases:

- (A left-inverting sequent $A * B, \Delta \longrightarrow C$): Any focused derivation must end in $*L$, so we apply the i.h. to $A, B, \Delta \longrightarrow C$.
- (A right-focusing sequent $\Gamma^{\mathrm{irr}} \longrightarrow A * B$): Any focused derivation must end in $*R$, and decide some splitting of the context into contiguous pieces Γ_1^{irr} and Δ_2. However, Γ_1^{irr} and Δ_2 are uniquely determined by frontier refinement ($\mathrm{fr}(\Gamma_1^{\mathrm{irr}}) = \mathrm{fr}(A)$ and $\mathrm{fr}(\Delta_2) = \mathrm{fr}(B)$) and the equation $\Gamma^{\mathrm{irr}} = \Gamma_1^{\mathrm{irr}}, \Delta_2$. So, we apply the i.h. to $\Gamma_1^{\mathrm{irr}} \longrightarrow A$ and $\Delta_2 \longrightarrow B$.
- (An atomic sequent $\Gamma^{\mathrm{irr}} \longrightarrow p$): The sequent has exactly one focused derivation if and only if $\Gamma^{\mathrm{irr}} = p$. ◀

Proof of Theorem 24. By Theorem 15 and Lemma 25. ◀

2.6 Notes

The coherence theorem says in a sense that focused derivations provide a canonical representation for intervals of the Tamari order. Although the representations are quite different, in this respect it seems roughly comparable to the "unicity of maximal chains" that was established by Tamari and Friedman as part of their original proof of the lattice property of Y_n [26, 6]. A natural question is whether the sequent calculus can be used to better understand and further simplify the proofs (cf. [8] [15, §4]) of this lattice property.

An easy observation is that one obtains the dual Tamari order (cf. Footnote 1) via a dual restriction of Lambek's original rule, in other words by requiring the product formula to appear at the *rightmost* end of the context. These two forms of product might also be considered in combination with left and right units, or in combination with Lambek's left and right division operations.[3] Interestingly, Lambek, who originally presented his "syntactic calculus" as a tool for mathematical linguistics, also introduced a fully non-associative version [11] (cf. [16, Ch. 4]); one may wonder whether there are any linguistic motivations for semi-associativity, as an intermediate point between these two extremes.

The name "coherence theorem" for Theorem 24 is inspired by the terminology from category theory and Mac Lane's coherence theorem for monoidal categories [13]. Laplaza [14] extended Mac Lane's coherence theorem to the situation (very close to Tamari's) where

[3] Since circulating an earlier version of this paper [31], I have learned that Jason Reed briefly considered precisely such an extension of the sequent calculus studied here (independently, of course), with left unit and right division [18]. Reed did not remark the connection with the Tamari order, but rather was interested in the apparent failure of the induced logic to satisfy Nuel Belnap's *display property*.

there is no monoidal unit and the *associator* $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$ is only a natural transformation rather than an isomorphism. (In the presence of units, this gives rise to the notion of a *skew-monoidal category* [24, 9], cf. [23].) The precise relationship with our proof-theoretic coherence theorem remains to be clarified.

3 Counting intervals in Tamari lattices

In this section we explain how the coherence theorem can be used to give a new proof of Chapoton's result (mentioned in the Introduction) that the number of intervals in Y_n is given by Tutte's formula (1) for planar triangulations. We will assume some basic familiarity with generating functions.

The problem of "counting intervals" is to compute the cardinality of the set $\mathcal{I}_n = \{ (A, B) \in \mathsf{Y}_n \times \mathsf{Y}_n \mid A \leq B \}$ as a function of n. By the soundness and completeness theorems as well as the frontier invariance property (Prop. 19), each Y_n is isomorphic as a partial order to the set of formulas A of size n with any fixed frontier of length $n+1$ (remember that a binary tree with n internal nodes has $n+1$ leaves), ordered by sequent derivability. By the coherence theorem, the problem of counting intervals can therefore be reduced to the problem of *counting focused derivations*.

This problem lends itself readily to being solved using generating functions. Consider the generating functions $L(z, x)$ and $R(z, x)$ defined as formal power series $L(z, x) = \sum_{k,n \in \mathbb{N}} \ell_{k,n} x^k z^n$ and $R(z, x) = \sum_{k,n \in \mathbb{N}} r_{k,n} x^k z^n$, where $\ell_{k,n}$ (respectively, $r_{k,n}$) is the number of focused derivations of sequents whose left-hand side is a context (respectively, irreducible context) of length k and whose right-hand side is a formula of size n. (Without loss of generality in this analysis, we assume that all formulas A of size n have a fixed frontier $\mathrm{fr}(A) = p^{n+1}$.) We write $L_1(z)$ to denote the coefficient of x^1 in $L(z, x)$.

▶ **Proposition 26.** *$L_1(z)$ is the ordinary generating function counting Tamari intervals by size.*

Proof. The coefficients $\ell_{1,n}$ give the number of focused derivations of sequents of the form $A \Longrightarrow B$, where $|B| = n$ (and hence $|A| = n$), so $\ell_{1,n} = |\mathcal{I}_n|$ by Theorem 24. ◀

▶ **Proposition 27.** *L and R satisfy the equations:*

$$L(z, x) = \frac{L(z, x) - xL_1(z)}{x} + R(z, x) \tag{2}$$

$$R(z, x) = zR(z, x)L(z, x) + x \tag{3}$$

Proof. The equations are derived directly from the inductive structure of focused derivations:

- The first summand in (2) corresponds to the contribution from the $*L$ rule, which transforms any $A, B, \Gamma \Longrightarrow C$ into $A * B, \Gamma \Longrightarrow C$. The context in the premise must have length ≥ 2 which is why we subtract the $xL_1(z)$ factor, and the context in the conclusion is one formula shorter which is why we divide by x. The second summand is the contribution from irreducible contexts.
- The first summand in (3) corresponds to the contribution from the $*R^{\mathrm{foc}}$ rule, which transforms $\Gamma^{\mathrm{irr}} \Longrightarrow A$ and $\Delta \Longrightarrow B$ into $\Gamma^{\mathrm{irr}}, \Delta \Longrightarrow A * B$: the length of the context in the conclusion is the sum of the lengths of Γ^{irr} and Δ, while the size of $A * B$ is one plus the sum of the sizes of A and B, which is why we multiply R and L together and then by an extra factor of z. The second summand is the contribution from $id^{\mathrm{atm}} : p \Longrightarrow p$. ◀

▶ **Proposition 28.** $L_1(z) = R(z, 1)$.

Proof. This follows algebraically from (2), but we can interpret it constructively as well. The coefficient of z^n in $R(z,1)$ is the formal sum $\sum_k r_{k,n}$, giving the number of focused derivations of sequents whose right-hand side is a formula of size n and whose left-hand side is an irreducible context of arbitrary length. But by Props. 21–23, the operations $\phi[-]$ and $\psi[-]$ realize a 1-to-1 correspondence between derivable sequents of the form $\Gamma^{\text{irr}} \longrightarrow B$ and ones of the form $A \longrightarrow B$. ◀

After substituting $L_1(z) = R(z,1)$ into (2) and applying a bit of algebra, we obtain another formula for L in terms of a "discrete difference operator" acting on R:

$$L(z,x) = x\frac{R(z,x) - R(z,1)}{x-1} \tag{4}$$

The recursive (or "functional") equations (3) and (4) can be easily unrolled using computer algebra software to compute the first few dozen coefficients of R and L:

$$R(z,x)=x+x^2z+(x^2+2x^3)z^2+(3x^2+5x^3+5x^4)z^3+(13x^2+20x^3+21x^4+14x^5)z^4+\ldots$$

$$L_1(z)=R(z,1)=1+z+3z^2+13z^3+68z^4+399z^5+2530z^6+16965z^7+\ldots$$

▶ **Theorem 29** (Chapoton [3]). $|\mathcal{I}_n| = \frac{2(4n+1)!}{(n+1)!(3n+2)!}$.

Proof. At this point, we can directly appeal to results of Cori and Schaeffer, because equations (3) and (4) are a special case of the functional equations given in [4] for the generating functions of *description trees of type* (a,b), where $a = b = 1$. Cori and Schaeffer explained how to solve these equations abstractly for $R(z,1)$ using Brown and Tutte's "quadratic method", and then how to derive the explicit formula above in the specific case that $a = b = 1$ via Lagrange inversion (essentially as the formula was originally derived by Tutte for planar triangulations). ◀

Let's take a moment to discuss Chapoton's original proof of Theorem 29, which it should be emphasized has many commonalities with the one given here, despite being less structured. Chapoton likewise defines a two-variable generating function $\Phi(z,x)$ enumerating intervals in the Tamari lattices Y_n, where the parameter z keeps track of n, and the parameter x keeps track of the *number of segments along the left border* of the tree at the lower end of the interval.[4] By a combinatorial analysis, Chapoton derives the following equation for Φ:

$$\Phi(z,x) = x^2z(1+\Phi(z,x)/x)\left(1+\frac{\Phi(z,x)-\Phi(z,1)}{x-1}\right) \tag{5}$$

He manipulates this equation and eventually solves for $\Phi(z,1)$ as the root of a certain polynomial, from which he derives Tutte's formula (1), again by appeal to another result in the paper by Cori and Schaeffer [4].

If we give a bit of thought to these definitions, it is easy to see that the number of segments along the left border of a tree (= formula) A is equal to the length of its maximal decomposition $\psi[A]$ – meaning that the generating function $\Phi(z,x)$ apparently contains exactly the same information as $R(z,x)$! There is a small technicality, however, due to the fact that Chapoton only considers the Y_n for $n \geq 1$. In fact, the two generating functions are related by a small offset (corresponding to the coefficient of z^0 in $R(z,x)$): $\Phi(z,x) = R(z,x) - x$. Indeed, it can be readily verified that equation (5) follows from (3) and (4), applying the substitution $R(z,x) = x + \Phi(z,x)$.

[4] Or rather at the upper end, since Chapoton uses the dual ordering convention (cf. Footnote 1).

Acknowledgment. I thank Wenjie Fang, Paul-André Melliès, Jason Reed, Gabriel Scherer, and Tarmo Uustalu for interesting conversations and helpful comments about this work. I also thank the anonymous FSCD reviewers for their generous readings of the paper and some useful remarks.

References

1 Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. Logic and Computation*, 2(3):297–347, 1992.
2 O. Bernardi and N. Bonichon. Intervals in Catalan lattices and realizers of triangulations. *J. Combin. Theory Ser. A*, 116(1):55–75, 2009.
3 F. Chapoton. Sur le nombre d'intervalles dans les treillis de Tamari. *Sém. Lothar. Combin.*, (B55f), 2006. 18 pp. (electronic).
4 Robert Cori and Gilles Schaeffer. Description trees and Tutte formulas. *Theoretical Computer Science*, 292(1):165–183, 2003. Selected papers in honor of Jean Berstel.
5 Wenjie Fang. Planar triangulations, bridgeless planar maps and Tamari intervals. arXiv:1611.07922, 2016.
6 H. Friedman and D. Tamari. Problèmes d'associativité: une structure de treillis finis induite par une loi demi-associative. *J. Combinatorial Theory*, 2:215–242, 1967.
7 Gerhard Gentzen. Investigations into logical deductions. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1969.
8 S. Huang and D. Tamari. Problems of associativity: A simple proof for the lattice property of systems ordered by a semi-associative law. *J. Combin. Theory Ser. A*, 13(1):7–13, 1972.
9 Stephen Lack and Ross Street. Triangulations, orientals, and skew monoidal categories. *Advances in Math.*, 258:351–396, 2014.
10 Joachim Lambek. The mathematics of sentence structure. *The American Mathematical Monthly*, 65(3):154–170, 1958.
11 Joachim Lambek. On the calculus of syntactic types. In R. Jakobson, editor, *Structure of Lang. and its Math. Aspects*, Proc. Symp. Appl. Math., XII, pages 166–178. AMS, 1961.
12 Sergei K. Lando and Alexander K. Zvonkin. *Graphs on Surfaces and Their Applications*. Number 141 in Encyclopaedia of Mathematical Sciences. Springer-Verlag, 2004.
13 Saunders Mac Lane. Natural associativity and commutativity. *Rice University Studies*, 49(4):28–46, 1963.
14 M. L. Laplaza. Coherence for associativity not an isomorphism. *J. Pure and Appl. Alg.*, 2(2):107–120, 1972.
15 Paul-André Melliès. Axiomatic rewriting theory VI: Residual theory revisited. In S. Tison, editor, *Rewriting Techniques and Appl.*, volume 2378 of *LNCS*, pages 24–50. Springer, 2002.
16 Richard Moot and Christian Retoré. *The Logic of Categorical Grammars: A Deductive Account of Natural Language Syntax and Semantics*, volume 6850 of *LNCS*. Springer, 2012.
17 F. Müller-Hoissen and H.-O. Walther, editors. *Associahedra, Tamari Lattices and Related Structures: Tamari Memorial Festschrift*, volume 299 of *Prog. in Math.* Birkhauser, 2012.
18 Jason Reed. Queue logic: An undisplayable logic? Unpublished manuscript, April 2009.
19 Jason Reed and Frank Pfenning. Focus-preserving embeddings of substructural logics in intuitionistic logic. Unpublished manuscript, January 2010.
20 D. D. Sleator, R. E. Tarjan, and W. P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. *J. Amer. Math. Soc.*, 1(3):647–681, 1988.
21 N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. `https://oeis.org`.
22 James Dillon Stasheff. Homotopy associativity of H-spaces, I. *Trans. Amer. Math. Soc.*, 108:293–312, 1963.

23 Ross Street. Skew-closed categories. *J. Pure and Appl. Alg.*, 217(6):973–988, 2013.
24 Kornél Szlachányi. Skew-monoidal categories and bialgebroids. *Advances in Math.*, 231(3–4):1694–1730, 2012.
25 Dov Tamari. *Monoïdes préordonnés et chaînes de Malcev.* Thèse, Université de Paris, 1951.
26 Dov Tamari. Sur quelques problèmes d'associativité. *Ann. sci. de Univ. de Clermont-Ferrand 2, Sér. Math.*, 24:91–107, 1964.
27 W. T. Tutte. A census of planar triangulations. *Canad. J. Math.*, 14:21–38, 1962.
28 T. R. S. Walsh and A. B. Lehman. Counting rooted maps by genus III: Nonseparable maps. *J. Combin. Theory Ser. B*, 18:222–259, 1975.
29 Noam Zeilberger. On the unity of duality. *Ann. Pure and Appl. Log.*, 153(1–3):66–96, 2008.
30 Noam Zeilberger. Linear lambda terms as invariants of rooted trivalent maps. *J. Functional Programming*, 26(e21), 2016.
31 Noam Zeilberger. A sequent calculus for the Tamari order. arXiv:1701.02917, 2017.
32 Noam Zeilberger and Alain Giorgetti. A correspondence between rooted planar maps and normal planar lambda terms. *Logical Methods in Comp. Sci.*, 11(3:22), 2015.