23rd International Conference on Rewriting Techniques and Applications

RTA'12, May 30–June 1, 2012, Nagoya, Japan

Edited by

Ashish Tiwari

LIPICS

Editor

Ashish Tiwari
Computer Science Laboratory
SRI International
Menlo Park
CA 94025, United States
`ashish.tiwari@sri.com`

ACM Classification 1998
D.1 Programming Techniques, D.2 Software Engineering, D.3 Programming Languages, 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, I.1 Symbolic and Algebraic Manipulation, I.2 Artificial Intelligence

ISBN 978-3-939897-38-5

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/978-3-939897-38-5.

Publication date
May, 2012

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 .

Digital Object Identifier: 10.4230/LIPIcs.RTA.2012.i

ISBN 978-3-939897-38-5 **ISSN 1868-8969** **http://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

ISSN 1868-8969

www.dagstuhl.de/lipics

Contents

[1] The author's present address is: Institut für Informatik, Universität Leipzig, Johannisgasse 26, D-04103 Leipzig, Germany

Preface

The 23rd International Conference on Rewriting Techniques and Applications (RTA 2012) was held from May 28 to June 2, 2012, in Nagoya, Japan. This proceedings volume contains the contributed papers and extended abstracts of invited talks presented at the conference.

RTA is the major forum for the presentation of research on all aspects of rewriting. Previous RTA conferences were held in Dijon (1985), Bordeaux (1987), Chapel Hill (1989), Como (1991), Montreal (1993), Kaiserslautern (1995), Rutgers (1996), Sitges (1997), Tsukuba (1998), Trento (1999), Norwich (2000), Utrecht (2001), Copenhagen (2002), Valencia (2003), Aachen (2004), Nara (2005), Seattle (2006), Paris (2007), Hagenberg/Linz (2008), Brasilia (2009), Edinburgh (2010), and Novi Sad (2011). RTA 2012 received 46 submissions (45 regular research papers and 1 system description), with contributing authors from 18 countries. We thank the authors of submitted papers for considering RTA for publishing their work.

The program committee consisted of 21 members from 8 countries. Each submitted paper was reviewed by at least 3 members of the Program Committee, with the help of 69 external reviewers (who provided a total of 74 reviews). After intense discussions and debates, a total of 22 regular research papers were accepted for publication. We thank all the members of the program committee, as well as all external reviewers for their hard work and critical comments.

On behalf of the Program Committee, we are happy to present the best paper award to Kazushige Terui for the paper "Semantic evaluation, intersection types and complexity of simply typed lambda calculus". In addition to the contributed papers, the RTA program also consisted of three invited talks: "Computational Real Algebraic Geometry in Practice" by Hirokazu Anai, "Rho-calculi for computation and logic" by Claude Kirchner, and "Dictionary-Based Tree Compression" by Sebastian Maneth. We thank the invited speakers for enriching the conference with their talks and participation.

The entire reviewing process–consisting of paper submission, review assignments, electronic Program Committee discussion, and preparation of the proceedings–was handled seamlessly by Andrei Voronkov's Easychair system. We thank Andrei Voronkov and the Easychair team for creating this invaluable tool and making it freely available.

Workshops associated with RTA 2012 included the Annual Meeting of the IFIP Working Group 1.6 on Term Rewriting, 1st International Workshop on Confluence (IWC 2012), 21st International Workshop on Functional and (Constraint) Logic Programming (WFLP 2012), 6th International Workshop on Higher-Order Rewriting (HOR 2012), and 1st International Workshop on Trends in Tree Automata and Tree Transducers (TTATT 2012). We thank the workshop organizers for enriching the technical program.

Many people helped make RTA 2012 a success. Our special thanks to Masahiko Sakai for doing an excellent job as the chair of the organizing committee. The organizing committee, consisting of Masahiko Sakai, Aart Middeldorp, Keiichirou Kusakari, Naoki Nishida and Nao Hirokawa, along with the rest of the local organization team, provided invaluable help in ensuring the success of the event.

The proceedings of RTA 2012 is being published as a volume in the LIPIcs series under a Creative Commons license, with free online access to all, and with authors retaining rights over their contributions. We wish to thank the editorial board of LIPIcs for agreeing to publish the current proceedings. Our special thanks to Marc Herbstritt from Schloss Dagstuhl, Leibniz Center for Informatics, for his very helpful and prompt support during production of the LIPIcs proceedings. We gratefully acknowledge his help and the help of

the entire team in the LIPIcs editorial office.

We acknowledge the finanicial support we received from our sponsors: the Kayamori Foundation of Informational Science Advancement, Support Center for Advanced Telecommunications Technology Research Foundation, and and the Research Foundation for the Electrotechnology of Chubu. We are also grateful to the Nagoya Convention and Visitors Bureau for helping in making local arrangements for the banquet, excursion, and local travel. Finally, we also thank the Nagoya University and the Computer Science Laboratory at SRI International for supporting us in our efforts to organize RTA 2012.

Ashish Tiwari
Menlo Park CA, April 2012

Conference Organization

Program Chair

Ashish Tiwari	SRI International

Conference Chair

Masahiko Sakai	Graduate School of Infomation Science, Nagoya University

Program Committee

Andreas Abel	LMU Munich
Zena Ariola	University of Oregon
Paolo Baldan	Dipartimento di Matematica Pura e Applicata, Universita' di Padova
Ahmed Bouajjani	LIAFA, University of Paris 7 (Paris Diderot)
Evelyne Contejean	LRI, CNRS, Univ Paris-Sud, Orsay
Irène Anne Durand	LaBRI Universit Bordeaux 1
Jörg Endrullis	Vrije Universiteit Amsterdam
Silvio Ghilardi	Dipartimento di Scienze dell Informazione, Universita degli Studi di Milano
Guillem Godoy	Universitat Politècnica de Catalunya (UPC)
Nao Hirokawa	Japan Advanced Institute of Science and Technology
Deepak Kapur	University of New Mexico
Jordi Levy	IIIA - CSIC
Paul-André Melliès	University of Paris 7
Pierre-Etienne Moreau	INRIA-LORIA Nancy
Joachim Niehren	INRIA Lille
Grigore Rosu	University of Illinois at Urbana-Champaign
Albert Rubio	Universitat Politècnica de Catalunya
Masahiko Sakai	Graduate School of Infomation Science, Nagoya University
Carolyn Talcott	SRI International
René Thiemann	University of Innsbruck
Ashish Tiwari	SRI International

Steering Committee

Salvador Lucas (Chair)	Valencia, Spain
Frederic Blanqui	INRIA, China
Ian Mackie	Palaiseau
Georg Moser (Publicity Chair)	Innsbruck, Austria
Joachim Niehren	INRIA Lille
Masahiko Sakai	Nagoya, Japan

RTA Web Page

`http://rewriting.loria.fr/rta/`

External Reviewers

Aubrun, Nathalie
Avanzini, Martin

Bagan, Guillaume
Bahr, Patrick
Bonfante, Guillaume

Chiba, Yuki
Ciobaca, Stefan
Cirstea, Horatiu
Corradini, Andrea
Cortier, Veronique
Courcelle, Bruno
Curien, Pierre-Louis

Damiani, Ferruccio
Danvy, Olivier
David, Claire
De Vries, Fer-Jan
Delaune, Stephanie
Diekert, Volker
Downen, Paul

Escobar, Santiago

Felgenhauer, Bertram
Fleury, Emmanuel

Gaboardi, Marco
Gadducci, Fabio
Gascon, Adria
Genet, Thomas
Grygiel, Katarzyna

Hanus, Michael
Heindel, Tobias
Hermant, Olivier
Holik, Lukas
Houtmann, Clement
Hoyrup, Mathieu

Kahrs, Stefan
Kesner, Delia
Koprowski, Adam
Kusakari, Keiichirou

Leroy, Xavier
Loader, Ralph
Lohrey, Markus

Meredith, Patrick
Mertens, Eric
Meseguer, Jose
Mitsuhashi, Ichiro
Moore, Brandon Michael
Moser, Georg

Nakata, Keiko
Ndione, Antoine
Nishida, Naoki

Ogawa, Mizuhito

Petit, Barbara

Ringeissen, Christophe
Roux, Cody

Salvati, Sylvain
Sambin, Giovanni
Sato, Haruhiko
Schmitz, Sylvain
Sebastian, Tom
Senizergues, Geraud
Serbanuta, Traian
Simonsen, Jakob Grue
Stefanescu, Andrei
Sternagel, Christian

Touili, Tayssir

van Raamsdonk, Femke
Villaret, Mateu

Winkler, Sarah

Yuen, Shoji

Zantema, Hans

Author Index

Computational Real Algebraic Geometry in Practice *

Hirokazu Anai[1]

1 **FUJITSU Laboratories Ltd / Kyushu University**
4-1-1 Kamikodanaka, Nakahara-ku, Kawasaki 211-8588, Japan
`h.anai@kyudai.jp`

Abstract

Real algebraic geometry deals with the solution set of (possibly quantified) systems of polynomial equations and/or inequalities over the real numbers, which arise frequently in science and engineering. Main concern in real algebraic geometry is to determine the properties of the solution sets such as non-emptiness, dimension and quantifier free description as a semi-algebraic set. Such tasks are carried out by symbolic and algebraic algorithms: *cylindrical algebraic decomposition (CAD)* or *quantifier elimination (QE)*. Various algorithms and deep complexity results about CAD and QE have been studied during the last several decades (see [1]). Moreover, practically efficient software systems of QE have been developed and also are applied to many nontrivial application problems (see [2, 3, 4, 5]). In this talk we explain several algorithms of CAD and QE together with their engineering applications.

1998 ACM Subject Classification I.1.2 [Symbolic and Algebraic Manipulation] Algorithms – Algebraic Algorithms

Keywords and phrases real algebraic geometry, quantifier elimination, cylindrical algebraic decomposition, symbolic optimization

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.1

Category Invited Talk

References

1 B.F. Caviness and J.R. Johnson (Eds.). Quantifier Elimination and Cylindrical Algebraic Decomposition. New York: Springer-Verlag, 1998.
2 T. Sturm. New Domains for Applied Quantifier Elimination. In Proceedings of the 14th International Workshop on Computer Algebra (CASC) 2006, pages 295-301, 2006.
3 N. Hyodo, M. Hong, H. Yanami, S. Hara and H. Anai. Solving and visualizing nonlinear parametric constraints in control based on quantifier elimination. Appl. Algebra Eng. Commun. Comput. 18(6): pp. 497–512, 2007.
4 T. Sturm and A. Tiwari. Verification and synthesis using real quantifier elimination. In Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation (ISSAC) 2011, pp. 329–336, 2011.
5 H. Iwane, H. Yanami and H. Anai. A Symbolic-Numeric Approach to Multi-Objective Optimization in Manufacturing Design. Mathematics in Computer Science 5(3): pp. 315–334, 2011.

* This work was partially supported by Grant-in-Aid for Scientific Research (B) 21340025 and the Global COE Program "Education-and-Research Hub for Mathematics-for-Industry" (Kyushu University).

Rho-Calculi for Computation and Logic

Claude Kirchner[1]

1 **Inria**
Domaine de Voluceau – Rocquencourt B.P. 105 – 78153 Le Chesnay, France
`claude.kirchner@inria.fr`

Abstract

The rho-calculi provide enlightening concepts for both computing and reasoning as well as their combination. They consist in the generalization of lambda-calculus to structures like terms, propositions or graphs and we will show how their interrelations with deduction provide powerful frameworks for the next generation of proof assistants.

1998 ACM Subject Classification F.4.1 Mathematical Logic – Lambda calculus and related systems, I.2.3 Deduction and Theorem Proving, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases Rewriting calculus, rho-calculus, deduction modulo

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.2

Category Invited Paper

It is reasonable to hope that the relationship between computation and mathematical logic will be as fruitful in the next century as that between analysis and physics in the last. The development of this relationship demands a concern for both applications and for mathematical elegance.

John McCarthy
A Basis for a Mathematical Theory of Computation, 1963

1 Structures for computation

When one wants to caracterize informatics as a science, one main concept is prominent: computation. When the computation of 2+2 leads to 4, this transformation can be described and performed in many different ways, for instance using lambda-calculus or using rewriting. If the structure of the information to be transformed is available, it is well known and widely used that this may result in a huge difference in the complexity of the computation: computing with Church numbers in the lambda-calculus is not the most efficient way to implement arithmetic. Rewriting is one of the universal computational model that allows the embedding of structures as a first class concept.

These structures can be very simple, yet quite useful, typically when just plain tree syntax is used. But they can be also quite elaborated, for example when satisfying some theory like associativity, commutativity, groups, list, arrays, arithmetic, . . .

To handle explicitly structures and fonctions, we defined the rho-calculus [11, 9] (also called rewriting calculus) and indeed a family of such calculi [12, 10, 23, 2, 18, 22, 1] to mention just a few. Related calculi should be mentioned like [21, 20].

These calculi, offering a natural integration of structures into the theory of functions and in particular in lambda-calculus, provide a useful background for rewriting theory, implementation and applications.

But they offer also the right background to base the logical integration of computation into deduction, a deep and non-trivial goal as emphasized by the quote of John McCarthy.

2 Computation for logic

The complementarity of computation and deduction is known and used since the rise of logic. It has not always been well understood or accepted, like in the now settled controversy about the use of computations in the proof of the fourth color theorem. A formal treatment of this combination, called deduction modulo and initiated in [15], allows us to define deduction systems like the sequent calculus or natural deduction modulo some congruence defined by a theory implemented by a computational system such as rewriting. The consequences of reasoning modulo are deep. Typically, cut elimination does no hold for any congruence but sufficient conditions on the theories can be given [16, 8] allowing one to deal modulo important theories [17, 14, 19]. The second fondamental consequence is that the deduction modulo logical systems allow proving exactly the same theorems as when integrating the modulo part in the theory, but the proof are extremely different. Indeed they can be in some cases arbitrary shorter as shown in [7, 6].

An interesting point is that the proof terms representing proofs in deduction modulo are rho-terms [5], thus providing a complete picture of the articulation between structures, computations and proofs.

3 Next steps

This strong relationship between the rho-calculi and logic opens many research and application tracks. One is related to the implementation of these concepts to design proof assistants where the users are freed of the computational steps of the proof they are conducting, in the lines of the prototypes `https://www.rocq.inria.fr/deducteam/Dedukti` [3] or `http://rho.loria.fr/lemuridae.html` [4]. The relationship between rho-calculi and rewriting logic [13] has to be deepened. Challenging open questions like unification and matching in rho-calculi will lead to better understanding of the automation of reasoning, opening new trends for implementations and applications. Finally the relationship between computation and deduction and in particular the relationship with proof complexity is a challenging open research area.

Acknowledgements I want to thank all the colleagues with whom I have been sharing these ideas and work on rho-calculi and deduction modulo and in particulier Horatiu Cirstea, Nachum Dershowitz, Gilles Dowek, Hélène Kirchner and Luigi Liquori for many enlightening discussions.

References

1 Paolo Baldan, Clara Bertolissi, Horatiu Cirstea, and Claude Kirchner. A rewriting calculus for cyclic higher-order term graphs. *Mathematical Structures in Computer Science*, 2006.

2 Clara Bertolissi. *The graph rewriting calculus : properties and expressive capabilities*. Thèse de doctorat, Institut National Polytechnique de Lorraine - INPL, Oct 2005.

3 Mathieu Boespflug. *Conception d'un noyau de vérification de preuves pour le lambda-Pi-calcul modulo*. PhD thesis, École Polytechnique, Palaiseau, January 2011.

4 Paul Brauner. *Fondements et mise-en-oeuvre de la Super Déduction Modulo*. Phd thesis, INPL, Jun 2010.

5 Paul Brauner, Clément Houtmann, and Claude Kirchner. Principles of superdeduction. In Luke Ong, editor, *Proceedings of LICS*, jul 2007.

6 Guillaume Burel. *Bonnes démonstrations en déduction modulo.* PhD thesis, Université Henri Poincaré (Nancy 1), 2009.

7 Guillaume Burel. Efficiently simulating higher-order arithmetic by a first-order theory modulo. *Logical Methods in Computer Science*, 7(1), 2011.

8 Guillaume Burel and Claude Kirchner. Cut elimination in deduction modulo by abstract completion. In Sergei Artemov and Anil Nerode, editors, *Proc. of the Symposium on Logical Foundations of Computer Science*, Lecture Notes in Computer Science, New-York, USA, jun 2007. Springer Verlag.

9 Horatiu Cirstea. *Rewriting Calculus: Foundations and Applications.* PhD thesis, Université Henri Poincaré - Nancy I, 2000.

10 Horatiu Cirstea, Germain Faure, and Claude Kirchner. A rho-calculus of explicit constraint application. In *Proceedings of the 5th workshop on rewriting logic and applications.* Electronic Notes in Theoretical Computer Science, 2004.

11 Horatiu Cirstea and Claude Kirchner. The rewriting calculus — Part I *and* II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9:427–498, May 2001.

12 Horatiu Cirstea, Claude Kirchner, and Luigi Liquori. Matching Power. In Aart Middeldorp, editor, *12th International Conference on Rewriting Techniques and Applications (RTA'2001)*, volume 2051 of *Lecture Notes in Computer Science*, pages 77–92, Utrecht (The Netherlands), May 2001. Springer. Also available as Technical Report A01-R-201, LORIA, Nancy (France).

13 Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science.* Springer, 2007.

14 Gilles Dowek, Thérèse Hardin, and Claude Kirchner. HOL-$\lambda\sigma$ an intentional first-order expression of higher-order logic. *Mathematical Structures in Computer Science*, 11(1):21–45, 2001.

15 Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31(1):33–72, Nov 2003.

16 Gilles Dowek and Benjamin Werner. Proof normalization modulo. *The Journal of Symbolic Logic*, 68(4):1289–1316, 2003.

17 Gilles Dowek and Benjamin Werner. Arithmetic as a theory modulo. In Jürgen Giesl, editor, *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture notes in Computer Science*, pages 423–437, 2005.

18 Germain Faure. *Structure et modèles de calcul de réécriture.* Thèse de doctorat, Université Henri Poincaré Nancy, 2007.

19 Olivier Hermant. Resolution is cut-free. *Journal of Automated Reasoning*, 44(3):245–276, 03 2010.

20 C. Barry Jay and Delia Kesner. First-class patterns. *J. Funct. Program.*, 19(2):191–225, 2009.

21 Jan Willem Klop, Vincent van Oostrom, and Roel C. de Vrijer. Lambda calculus with patterns. *Theor. Comput. Sci.*, 398(1-3):16–31, 2008.

22 Luigi Liquori and Bernard P. Serpette. irho: an imperative rewriting calculus. In *6th Conference on Principles and Practice of Declarative Programming – PPDP'04*, pages 167–178. ACM, 2004.

23 Benjamin Wack. *Typage et déduction dans le calcul de réécriture.* Thèse de doctorat, Université Henri Poincaré - Nancy I, October, 7- 2005.

Dictionary-Based Tree Compression

Sebastian Maneth[*1]

1 **NICTA and University of New South Wales**
Sydney, Australia
`sebastian.maneth@nicta.com.au`

Abstract

Trees are a ubiquitous data structure in computer science. LISP, for instance, was designed to manipulate nested lists, that is, ordered unranked trees. Already at that time, DAGs were used to detect common subexpression, a process known as "hash consing." In a DAG every distinct subtree is represented only once (but can be referenced many times) and hence it constitutes a dictionary-based compression method for ordered trees. In our compression scenario we distinguish two kinds of ordered trees: binary and unranked. The latter appear naturally as representation of XML document structures. We survey these dictionary-based compression methods for ordered trees:

1. DAGs
2. hybrid DAGs
3. straight-line context-free tree grammars ("SLT grammars").

We compare the minimal DAG of an unranked tree with the minimal DAG of its binary tree encoding. The latter is obtained by identifying first children of the unranked tree with left children of the binary tree, and next-siblings with the right children. For XML document trees, unranked DAGs are usually smaller than encoded binary DAGs. We show that this holds for arbitrary unranked trees, *on average*. We also present the "hybrid DAG"; its size lower-bounds those of the binary and unranked DAGs.

Finding a smallest SLT grammar for a given tree is NP-complete. We discuss two linear-time approximation algorithms: BPLEX and TreeRePair. For typical XML document trees, TreeRePair produces SLT grammars that are only one fourth of the size of the minimal DAG, and which contain approximately 3% of the edges of the original tree. As far as we know, this gives rise to the smallest existing pointer-based tree representation.

We show that some basic algorithms can be computed directly on the compressed trees, without prior decompression. Examples include the execution of different kinds of tree automata, and the real-time traversal of the original tree. It is even possible to evaluate simple XPath queries directly on the SLT grammars, using deterministic node-selecting tree automata. In this way, impressive speed-ups are achieved over existing XPath evaluators, while at the same time the memory requirement is slashed to only a few percent. For more complex XPath queries that require nondeterministic node-selecting tree automata, efficient evaluation over SLT grammars remains a difficult challenge.

1998 ACM Subject Classification E.4 Data Compaction and Compression

Keywords and phrases Tree grammars, tree automata, straight-line programs

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.5

Category Invited Talk

[*] The author's present address is: Institut für Informatik, Universität Leipzig, Johannisgasse 26, D-04103 Leipzig, Germany

An Abstract Factorization Theorem for Explicit Substitutions

Beniamino Accattoli

INRIA & LIX (École Polytechnique), France
beniamino.accattoli@inria.fr

Abstract

We study a simple form of standardization, here called factorization, for explicit substitutions calculi, i.e. lambda-calculi where beta-reduction is decomposed in various rules. These calculi, despite being non-terminating and non-orthogonal, have a key feature: each rule terminates when considered separately. It is well-known that the study of rewriting properties simplifies in presence of termination (e.g. confluence reduces to local confluence). This remark is exploited to develop an abstract theorem deducing factorization from some axioms on local diagrams. The axioms are simple and easy to check, in particular they do not mention residuals. The abstract theorem is then applied to some explicit substitution calculi related to Proof-Nets. We show how to recover standardization by levels, we model both call-by-name and call-by-value calculi and we characterize linear head reduction via a factorization theorem for a linear calculus of substitutions.

1998 ACM Subject Classification F.4.1 Mathematical Logic – Lambda calculus and related systems, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases λ-calculus, Standardization, Explicit Substitutions, Abstract rewriting, Diagrammatic reasoning

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.6

Category Regular Research Paper

1 Introduction

Background. The study of the rewriting properties of a system S simplifies considerably if there cannot be infinite reductions, *i.e.* if the rewriting relation $\to_S$ associated to S is strongly normalizing. The typical example is the study of confluence, which can be reduced to local confluence in presence of strong normalization, thanks to Newman's lemma. Unfortunately, many interesting systems—λ-calculus for instance—do not enjoy strong normalization ($\mathcal{SN}$)[1].

There is a special class of non strongly normalizing rewriting systems which is given by those systems having more than one rewriting rule and s.t. any of their rules is $\mathcal{SN}$ when considered alone; they could be called *locally terminating systems*. Natural examples are most λ-calculi where β-reduction is decomposed into a set of more atomic rules. This is typically done by extending the syntax of λ-calculus with a new term constructor $t[x/u]$ (sometimes written `let` $x = u$ `in` t) denoting a delayed or **explicit substitution** (ES), and decomposing the β-rule:

$$(\lambda x.t)\ u \to_\beta t\{x/u\}$$

[1] λ-calculus is an orthogonal system, and so confluence of λ-calculus is easy for other reasons. But the systems we will deal with in this paper are not orthogonal.

into two rules:

$$(\lambda x.t)\ u \rightarrow_{\mathtt{B}} t[x/u] \rightarrow_{\mathtt{s}} t\{x/u\}$$

In the ES-literature $\rightarrow_{\mathtt{s}}$ is in turn decomposed into a set of rules (see [22] for a survey), but for the present discussion this is not relevant. Explicit substitutions were introduced to close the gap between the theory of λ-calculus and implementations [1]. Their rewriting theory has also been the subject of a lot of research, after Melliès showed the possibility of pathological behaviors [25].

A similar decomposition of $\rightarrow_\beta$ arise in many contexts, not necessarily in relation to implementations and often without explicit reference to ES. Indeed, it can be found in studies on call-by-value [19], complexity classes [32], linear systems [21], side-effects [20], etc. Usually, these refined systems are not orthogonal nor strongly normalizing, and their rewriting theory may be involved.

Local to global, via $\mathcal{SN}$. In locally terminating systems termination of each rule can sometimes be used to reduce *global* rewriting properties to a *local* form, similarly to how Newman's lemma reduces confluence to local confluence. The aim of the paper is to reduce a simple form of standardization for ES-calculi to a local form, using as key ingredient termination of each single rule.

Termination and confluence concern the *existence* and the *uniqueness* of normal forms, which are the results of a computation. Standardization instead is about *how to compute*: it identifies a specific class of reductions to which any other reduction can be transformed by permuting its steps. It has many important corollaries, in particular it gives a normalizing strategy for evaluation (see [9]).

Factorization in λ-calculus. It is well-known that in λ-calculus any reduction $t \rightarrow_\beta^* u$ can be re-organised so that

$$t \rightarrow_{\mathtt{h}}^* \rightarrow_{\mathtt{i}}^* u$$

i.e. by first reducing on the head ($\rightarrow_{\mathtt{h}}^*$) and then everywhere else ($\rightarrow_{\mathtt{i}}^*$). Concisely this is expressed by $\rightarrow_\beta^* = (\rightarrow_{\mathtt{h}} \cup \rightarrow_{\mathtt{i}})^* \subseteq \rightarrow_{\mathtt{h}}^* \rightarrow_{\mathtt{i}}^*$. This is not exactly the standardization theorem: it is an easier *factorization* theorem which can be inferred from standardization [27] or it can be used to prove it [30, 9]. "Conceptually, the factorisation property means that the efficient part of a computation can always be separated from its junk" [27]. Applications of standardization often only require factorization, *e.g.* the operational characterization of solvability for λ-calculus ([9]).

At first sight factorization is easy: a two-steps sequence as $\rightarrow_{\mathtt{i}} \rightarrow_{\mathtt{h}}$ can always be permuted, and the theorem seems to follow from this local permutation property. The swap is indeed possible, but it is non-linear, *i.e.* it has the following general form $\rightarrow_{\mathtt{i}} \rightarrow_{\mathtt{h}} \subseteq \rightarrow_{\mathtt{h}}^+ \rightarrow_{\mathtt{i}}^*$, which can diagrammatically be represented as:

$$\begin{tikzcd}
t \arrow[r, "\mathtt{i}"] \arrow[d, dashed, "\mathtt{h}+"'] \arrow[dr, phantom, "\Swarrow"] & u \arrow[d, "\mathtt{h}"] \\
v \arrow[r, dashed, "\mathtt{i}^*"'] & w
\end{tikzcd}$$

Now, there are two easy abstract lemmas (Lemma 4.2 and 4.3), similar to Newman's lemma, which would imply the factorization theorem *but only when* either the sequence $\rightarrow_{\mathtt{h}}^+$ is composed of at most one step or $\rightarrow_{\mathtt{h}}$ is strongly normalizing. Unfortunately, neither of these conditions holds for λ-calculus. Indeed, consider the following diagram (where $I = \lambda y.y$):

$$\begin{tikzcd}
(\lambda x.x\ x)\ (I\ I) \arrow[rr, "\mathtt{i}"] \arrow[d, dashed, "\mathtt{h}"] & \arrow[d, phantom, "\Swarrow"] & (\lambda x.x\ x)\ I \arrow[d, "\mathtt{h}"] \\
(I\ I)\ (I\ I) \arrow[r, dashed, "\mathtt{h}"'] & I\ (I\ I) \arrow[r, dashed, "\mathtt{i}"'] & I\ I
\end{tikzcd}$$

It shows that in general $\to_{\mathtt{h}}^{+}$ can have length greater than one[2]. And there exist $\to_{\mathtt{h}}$-divergent terms, for instance $(\lambda x.(x\ x))\ \lambda x.(x\ x)$. Therefore, factorization in non-terminating systems is non-trivial.

The rewriting technique. Let us sketch the idea from which this paper stems. The decomposition of β into $\mathtt{B}$ and $\mathtt{s}$ induces head (noted $\to_{\mathtt{B}\circ}$ and $\to_{\mathtt{s}\circ}$) and internal ($\to_{\mathtt{B}\bullet}$ and $\to_{\mathtt{s}\bullet}$) variants of $\mathtt{B}$ and $\mathtt{s}$. The new head reduction $\to_{\circ}$, given by $\to_{\circ}:=\to_{\mathtt{B}\circ} \cup \to_{\mathtt{s}\circ}$, is not strongly normalizing, but now it splits into two strongly normalizing reductions, $\to_{\mathtt{B}\circ}$ and $\to_{\mathtt{s}\circ}$. The internal-head permutation splits into four diagrams (the dotted lines are not specified on purpose):

$$\begin{tikzcd} t \arrow[r, "\mathtt{B}\bullet"] \arrow[d, dashed, "*"'] \arrow[rd, phantom, "\Leftarrow"] & u \arrow[d, "\mathtt{B}\circ"] \\ v \arrow[r, dashed, "*"] & w \end{tikzcd} \quad \begin{tikzcd} t \arrow[r, "\mathtt{s}\bullet"] \arrow[d, dashed, "*"'] \arrow[rd, phantom, "\Leftarrow"] & u \arrow[d, "\mathtt{s}\circ"] \\ v \arrow[r, dashed, "*"] & w \end{tikzcd} \quad \begin{tikzcd} t \arrow[r, "\mathtt{B}\bullet"] \arrow[d, dashed, "*"'] \arrow[rd, phantom, "\Leftarrow"] & u \arrow[d, "\mathtt{s}\circ"] \\ v \arrow[r, dashed, "*"] & w \end{tikzcd} \quad \begin{tikzcd} t \arrow[r, "\mathtt{s}\bullet"] \arrow[d, dashed, "*"'] \arrow[rd, phantom, "\Leftarrow"] & u \arrow[d, "\mathtt{B}\circ"] \\ v \arrow[r, dashed, "*"] & w \end{tikzcd}$$

These diagrams can now be studied one by one using strong normalization of $\to_{\mathtt{B}\circ}$ and $\to_{\mathtt{s}\circ}$, which allows to apply the abstract lemmas (Lemma 4.2 and 4.3, and actually a new abstract lemma, Lemma 4.4, is also needed). The obtained factorizations can then easily be glued together, getting the factorization theorem for $\to_{\circ}$. Therefore factorization—surprisingly—turns out to be easier in ES-calculi than in λ-calculus. Actually, in λ-calculus it is possible to prove factorization using some termination assumptions, by considering complete developments of given sets of redexes, which are always finite (see [9], Chapter 11.4). However, this technique requires to define complex notions such as residuals and developments, while our approach circumvents their need.

The main result of the paper is a set of axioms on local diagrams which imply the factorization theorem. Despite being conceived for ES-calculi the axiomatization does not rely on any particular feature of such calculi, it is simply formulated in terms of swaps between reduction relations. Moreover, the axioms are simple and can be checked without using heavy techniques as positions, labels or underlinings. Last, the theorem follows from the axioms quite easily, once the right abstract lemmas for postponement have been isolated.

Applications. We developed the abstract result of this paper while studying standardization for some λ-calculi related to Linear Logic Proof-Nets. Starting from these graphical syntaxes, a new *at a distance* approach to explicit substitutions has recently been proposed [5]. Distance-calculi are simpler than those of the earlier generation, in particular they have less rules and they led to new results on λ-calculus [5, 6]. We present various concrete applications of our abstract theorem to three such ES-calculi:

1. **The substitution calculus** $\lambda_{\mathtt{sub}}$: the head reduction of $\lambda_{\mathtt{sub}}$ corresponds to reduction at level 0 in Proof-Nets. We prove factorization and show how to iterate it in order to get a standardization theorem by levels in the style of Linear Logic Proof-Nets.
2. **The value substitution calculus** $\lambda_{\mathtt{vsub}}$, a call-by-value version of $\lambda_{\mathtt{sub}}$. We factorize $\lambda_{\mathtt{vsub}}$ reduction with respect to weak reduction, which plays the same role as head reduction in the call-by-name case. We also factorize $\lambda_{\mathtt{vsub}}$ reduction with respect to another scheme, obtained by mixing head and weak reduction. These factorizations have been used in [8] to give a new characterization of call-by-value solvability.
3. **The linear substitution λ-calculus** $\lambda_{\mathtt{lsub}}$: it is a a natural refinement of $\lambda_{\mathtt{sub}}$ where explicit substitutions act on one occurrence at the time. It is also a slight variation over

[2] It is enough to replace $\lambda x.x\ x$ with $\lambda x.(x\ x)\ x$ in the example to get that also $\to_{\mathtt{i}}^{*}$ has length greater than one

a calculus proposed by Robin Milner (and related to Bigraphs and Pure Proof-Nets), and a simplification of the structural calculus $\lambda\mathtt{j}$ of [5]. Surprisingly, the reductions by levels of $\lambda_{\mathtt{sub}}$ do not induce factorization nor standardization for $\lambda_{\mathtt{lsub}}$. That approach is then refined: the proper head reduction of $\lambda_{\mathtt{lsub}}$ is head *linear* reduction, an important notion having strong connections with Krivine's abstract machine (KAM) [14], Proof-Nets [23], Geometry of Interaction [13], Game Semantics [12], Differential λ-calculus [16] and the translation of λ-calculus into the π-calculus [24]. Ours is the first characterization of linear head reduction from a rewriting point of view. Moreover, we generalize linear head reduction into an indexed family of linear reductions, and give a generalized factorization theorem for $\lambda_{\mathtt{lsub}}$.

Related work. Standardization was first studied axiomatically by Gonthier, Levy and Melliès in [18], and then further explored by Melliès [26, 28], who also gives axioms for factorization in [27]. Our approach differs in two main points: 1) his theory is very general and powerful, but also complex and difficult to apply, while ours, having been designed around a specific class of systems, is easy to apply, but of course its scope is limited; 2) we use factorization to prove standardization while he proceeds the other way around.

Road-map. In Section 2 we fix some terminology and notation. In Section 3 we define the substitution calculus $\lambda_{\mathtt{sub}}$ and use it to introduce our axiomatics. Section 4 recalls and develops some abstract tools about postponement of reductions. In Section 5 any system satisfying the axioms is shown to enjoy a factorization theorem. In Section 6 we prove two factorizations for the value-substitution calculus $\lambda_{\mathtt{vsub}}$. Section 7 introduces the linear substitution calculus $\lambda_{\mathtt{lsub}}$, factorization with respect to linear head reduction, and then generalizes linear head reduction. Section 8 concludes.

2 Notations and conventions

We follow a relational approach to abstract rewriting, *i.e.* we see a rewriting rule $\to$ on a set S as a subset of $\to \subseteq S \times S$. Composition of rewriting steps is simple juxtaposition, therefore $\to_1\to_2$ stands for a sequence like $t \to_1 u \to_2 v$. In the following we use $\to_1\to_2 \subseteq \to_3\to_4$ as a compact notation for a diagram like:

$$\begin{tikzcd} t \arrow[r, "1"] \arrow[d, dashed, "3"'] & u \arrow[d, "2"] \arrow[dl, phantom, "\Swarrow"] \\ v \arrow[r, dashed, "4"'] & w \end{tikzcd}$$

Composition of rewriting rules has precedence over union, *i.e.*$\to_1\to_2 \cup \to_3$ should be read as $(\to_1\to_2)\cup \to_3$. Given a rewriting relation $\to$ we note $\to^+$ and $\to^*$ the transitive and transitive-reflexive closure of $\to$, respectively. We will note $\to_\circ$ and $\to_\bullet$ the head and internal variant of a given relation $\to$. Unfortunately, the paper requires a huge number of rewriting relations.

3 The substitution calculus $\lambda_{\mathtt{sub}}$

The calculus $\lambda_{\mathtt{sub}}$ is given by the following syntax for terms:

$$t, u, s, v, r ::= x \mid \lambda x.t \mid t\ u \mid t[x/u]$$

The constructor $t[x/u]$ is called an **explicit substitution**. Both $\lambda x.t$ and $t[x/u]$ bind x in t. We use $\mathtt{L}$ to note a possibly empty list of explicit substitutions $[x_1/u_1] \ldots [x_k/u_k]$. The rewriting relation $\rightarrow_{\lambda_{\mathtt{sub}}}$ of $\lambda_{\mathtt{sub}}$ is given by the union of rules $\rightarrow_{\mathtt{dB}}$ and $\rightarrow_{\mathtt{s}}$, which are obtained as the context closure of following two root rules:

$$(\lambda x.t)\mathtt{L}\ s \quad \mapsto_{\mathtt{dB}} \quad t[x/s]\mathtt{L} \qquad\qquad t[x/u] \quad \mapsto_{\mathtt{s}} \quad t\{x/u\}$$

Rule $\rightarrow_{\mathtt{dB}}$ is the rule labeled $\mathtt{B}$ at page 1 but here acting *at a distance* (which is the reason for $\mathtt{d}$): the function $\lambda x.t$ and the argument s can interact even if there is $\mathtt{L}$ between them. This is motivated by the close relation between $\lambda_{\mathtt{sub}}$ and graphical formalisms as (Pure) Proof-Nets or $\lambda\mathtt{j}$-dags, see [3, 4]. Morally, $\rightarrow_{\mathtt{dB}}$ (resp. $\rightarrow_{\mathtt{s}}$) corresponds to the multiplicative (resp. exponential) reduction of Proof-Nets. With different motivations this rule was also considered by Curien, Hardin and Lévy in [11].

The substitution calculus may be suspected to be a useless modification of the λ-calculus, because it has explicit substitutions but it executes them in just one shot. However, in [6] $\lambda_{\mathtt{sub}}$ is the crucial tool in the proofs of some results on λ-calculus for which no other proof is known. It should be taken as a tool to study λ-calculus rather than implementations.

The reduction relation $\rightarrow_{\lambda_{\mathtt{sub}}}$ of $\lambda_{\mathtt{sub}}$ is not $\mathcal{SN}$, since $\rightarrow_\beta\subseteq\rightarrow_{\lambda_{\mathtt{sub}}}\rightarrow_{\lambda_{\mathtt{sub}}}$. However, both $\rightarrow_{\mathtt{dB}}$ and $\rightarrow_{\mathtt{s}}$ are $\mathcal{SN}$.

▶ **Lemma 3.1** ([6]). *The reductions $\rightarrow_{\mathtt{dB}}$ and $\rightarrow_{\mathtt{s}}$ are strongly normalizing and confluent. Moreover, $\rightarrow_{\lambda_{\mathtt{sub}}}$ is confluent.*

The literature on λ-calculus usually factors β reduction with respect to head and internal reductions. We are going to do the same in $\lambda_{\mathtt{sub}}$, but our notion of head reduction will be slightly more liberal than the usual one. **Head contexts** H are defined by:

$$H \quad ::= \quad [\cdot] \mid \lambda x.H \mid H\ t \mid H[x/t]$$

The reductions $\rightarrow_{\mathtt{dB}\circ}$ and $\rightarrow_{\mathtt{s}\circ}$ are defined as the closure by *head* contexts of $\mapsto_{\mathtt{dB}}$ and $\mapsto_{\mathtt{s}}$. Then $\lambda_{\mathtt{sub}}$ head reduction is noted $\rightarrow_{\lambda_{\mathtt{sub}}\circ}$ and given by $\rightarrow_{\mathtt{dB}\circ} \cup \rightarrow_{\mathtt{s}\circ}$.

Our head reduction is a non-deterministic strategy. Consider:

$$(\lambda x.y[y/I])\ u \quad {}_{\mathtt{dB}}\!\leftarrow \quad (\lambda x.(I\ I))\ u \quad \rightarrow_{\mathtt{dB}} \quad (I\ I)[x/u]$$

However, a simple case analysis shows that $\rightarrow_{\lambda_{\mathtt{sub}}\circ}$ enjoys the diamond property, which is just a more abstract way to say that its non-determinism is harmless: it does not affect reduction lengths and $\rightarrow_{\lambda_{\mathtt{sub}}\circ}$ normalizes weakly iff it normalizes strongly; in fact $\rightarrow_{\lambda_{\mathtt{sub}}\circ}$ can be considered as a deterministic strategy. In particular, reducing $\rightarrow_{\lambda_{\mathtt{sub}}\circ}$ in an outermost way recovers a more traditional formulation of head reduction. A motivation for our approach is that $\rightarrow_{\lambda_{\mathtt{sub}}\circ}$ corresponds exactly to reduction at level 0 in Linear Logic Proof-Nets.

In order to study factorization with respect to head reduction we need to define the complements of $\rightarrow_{\lambda_{\mathtt{sub}}\circ}$,$\rightarrow_{\mathtt{dB}\circ}$ and $\rightarrow_{\mathtt{s}\circ}$. We define the **internal reduction** $\rightarrow_{\lambda_{\mathtt{sub}}\bullet}$ as the union of $\rightarrow_{\mathtt{dB}\bullet}$ and $\rightarrow_{\mathtt{s}\bullet}$, where $\rightarrow_{\mathtt{dB}\bullet}:=\rightarrow_{\mathtt{dB}} \setminus \rightarrow_{\mathtt{dB}\circ}$ and $\rightarrow_{\mathtt{s}\bullet}:=\rightarrow_{\mathtt{s}} \setminus \rightarrow_{\mathtt{s}\circ}$. It is possible to show that $\rightarrow_{\mathtt{dB}\bullet}$ and $\rightarrow_{\mathtt{s}\bullet}$ can be defined as the closure by internal contexts of $\mapsto_{\mathtt{dB}}$ and $\mapsto_{\mathtt{s}}$, where **internal contexts** I are defined by:

$$I \quad ::= \quad t\ C \mid t\ [x/C] \mid H[I]$$

where C is no matter which context (and the notation $H[I]$ denotes the context obtained substituting the hole of H with I). The reductions are summarized by the square in Fig. 1.a, where each coordinate is equal to the union of the row/column it denotes.

The local internal-head permutations are given by the following lemma, proved by induction on the internal reductions and by means of a substitution lemma (details in [2]):

a)

	$\to_{\lambda_{\mathtt{sub}}\bullet}$	$\to_{\lambda_{\mathtt{sub}}\circ}$
$\to_{\mathtt{dB}}$	$\to_{\mathtt{dB}\bullet}$	$\to_{\mathtt{dB}\circ}$
$\to_{\mathtt{s}}$	$\to_{\mathtt{s}\bullet}$	$\to_{\mathtt{s}\circ}$

b)

	$\to_{\bullet}$	$\to_{\circ}$
$\rightsquigarrow$	$\rightsquigarrow_{\bullet}$	$\rightsquigarrow_{\circ}$
$\rightarrowtail$	$\rightarrowtail_{\bullet}$	$\rightarrowtail_{\circ}$

Figure 1 Squares of reductions

▶ **Lemma 3.2** (head-internal diagrams). *The following swaps hold:*

1. $\to_{\mathtt{dB}\bullet}\to_{\mathtt{dB}\circ}\subseteq\to_{\mathtt{dB}\circ}\to_{\mathtt{dB}\bullet}$.

2. $\to_{\mathtt{s}\bullet}\to_{\mathtt{s}\circ}\subseteq\to_{\mathtt{s}\circ}\to_{\mathtt{s}\circ}\to^{*}_{\mathtt{s}\bullet}\cup\to_{\mathtt{s}\circ}\to^{*}_{\mathtt{s}\bullet}$.

3. $\to_{\mathtt{s}\bullet}\to_{\mathtt{dB}\circ}\subseteq\to_{\mathtt{dB}\circ}\to_{\mathtt{s}\bullet}$.

4. $\to_{\mathtt{dB}\bullet}\to_{\mathtt{s}\circ}\subseteq\to_{\mathtt{s}\circ}\to_{\mathtt{dB}\circ}\to^{*}_{\mathtt{dB}\bullet}\cup\to_{\mathtt{s}\circ}\to^{*}_{\mathtt{dB}\bullet}$.

Points 1-3 give swaps for which we can apply some abstract lemmas in the literature, since $\to_{\mathtt{dB}\circ}$ and $\to_{\mathtt{s}\circ}$ are strongly normalizing, and easily get factorization. However, Point 4 introduces a new behaviour: swapping $\to_{\mathtt{dB}\bullet}\to_{\mathtt{s}\circ}$ can yield $\to_{\mathtt{s}\circ}\to_{\mathtt{dB}\circ}\to^{*}_{\mathtt{dB}\bullet}$, *i.e.* a third kind of reduction ($\to_{\mathtt{dB}\circ}$) can pop out. Let us give an example:

$$\begin{tikzcd}
(x\ x)[x/(\lambda z.z)\ y] \arrow[d, dashed, "\mathtt{s}\circ"] \arrow[rr, "\mathtt{dB}\bullet"] & \Swarrow & (x\ x)[x/z[z/y]] \arrow[d, "\mathtt{s}\circ"] \\
((\lambda z.z)\ y)\ ((\lambda z.z)\ y) \arrow[r, dashed, "\mathtt{dB}\circ"] & z[z/y]\ ((\lambda z.z)\ y) \arrow[r, dashed, "\mathtt{dB}\bullet"] & z[z/y]\ z[z/y]
\end{tikzcd}$$

Note that the diagram can be written as $\to_{\mathtt{dB}\bullet}\to_{\mathtt{s}\circ}\subseteq\to_{\mathtt{s}\circ}\to^{*}_{\mathtt{dB}}$. This phenomenon cannot happen in λ-calculus, where there is only one head and one internal rewriting relation. Other explicit substitution calculi (e.g. those in Section 6 and 7) present similar (but not identical) forms of permutations. They can be grouped together under the notion of *square factorization system.*

Let us abstract $\to_{\mathtt{dB}}$ and $\to_{\mathtt{s}}$ as two reductions $\rightsquigarrow$ and $\rightarrowtail$. Now, the square of reductions becomes as in Fig. 1.b, where we use the notations $\to_{\circ}:=\rightsquigarrow_{\circ}\cup\rightarrowtail_{\circ}$, $\to_{\bullet}:=\rightsquigarrow_{\bullet}\cup\rightarrowtail_{\bullet}$. Let us also set $\to:=\to_{\circ}\cup\to_{\bullet}=\rightsquigarrow\cup\rightarrowtail$.

▶ **Definition 3.3.** A **square factorization system** (SFS) S is given by a set $|S|$ and four reduction relations $(\rightsquigarrow_{\bullet},\rightsquigarrow_{\circ},\rightarrowtail_{\bullet},\rightarrowtail_{\circ})$ on $|S|$ s. t. the following conditions hold:

1. **Termination**: $\rightsquigarrow_{\circ}$ and $\rightarrowtail_{\circ}$ are strongly normalizing.
2. **Row-swap 1**: $\rightsquigarrow_{\bullet}\rightsquigarrow_{\circ}\subseteq\rightsquigarrow^{+}_{\circ}\rightsquigarrow^{*}_{\bullet}$.
3. **Row-swap 2**: $\rightarrowtail_{\bullet}\rightarrowtail_{\circ}\subseteq\rightarrowtail^{+}_{\circ}\rightarrowtail^{*}_{\bullet}$.
4. **Diagonal-swap 1**: $\rightarrowtail_{\bullet}\rightsquigarrow_{\circ}\subseteq\rightsquigarrow_{\circ}\rightarrowtail^{*}$.
5. **Diagonal-swap 2**: $\rightsquigarrow_{\bullet}\rightarrowtail_{\circ}\subseteq\rightarrowtail_{\circ}\rightsquigarrow^{*}$.

The names of axioms 2-5 refer to the square table in Fig. 1.b.

Note the presence of $\rightarrowtail=\rightarrowtail_{\circ}\cup\rightarrowtail_{\bullet}$ in axiom 4 and $\rightsquigarrow=\rightsquigarrow_{\circ}\cup\rightsquigarrow_{\bullet}$ in axiom 5: the diagonal swaps are where a third reduction pops out. Interestingly, the axioms are symmetric, *i.e.* swapping the roles of $\rightarrowtail$ and $\rightsquigarrow$ one still has a SFS.

In Section 5 we prove that any SFS enjoys a factorization theorem, *i.e.* $\to^{*}\subseteq\to^{*}_{\circ}\to^{*}_{\bullet}$ holds for any SFS. The important assumption in order to handle the diagonal swaps is that the anticipated relation (e.g. $\rightsquigarrow_{\circ}$ in axiom 4) occurs only once in the permuted reduction. Now, lemmas 3.2 and 3.1 prove that $(\to_{\mathtt{dB}\bullet},\to_{\mathtt{dB}\circ},\to_{\mathtt{s}\bullet},\to_{\mathtt{s}\circ})$ is a SFS, therefore by the abstract factorization result in Section 5 (Theorem 5.2.2, page 15) it follows:

▶ **Corollary 3.4.** $\to^{*}_{\lambda_{\mathtt{sub}}}\subseteq\to^{*}_{\lambda_{\mathtt{sub}}\circ}\to^{*}_{\lambda_{\mathtt{sub}}\bullet}$.

It is natural to wonder if from factorization for $\lambda_{\mathtt{sub}}$ one can recover the usual factorization for λ-calculus. Let us sketch an aswer. To every $\lambda_{\mathtt{sub}}$ term t one can associate the λ-term $\mathtt{s}(t)$, which is the normal form with respect to $\rightarrow_{\mathtt{s}}$ and it is easy to show that if $t \rightarrow_{\lambda_{\mathtt{sub}}} t'$ then $\mathtt{s}(t) \rightarrow_{\beta}^{*} \mathtt{s}(t')$. Such a projection does not map factorized reductions in $\lambda_{\mathtt{sub}}$ to factorized reduction in λ-calculus, because internal steps may project on head steps . Consider for instance the reduction $t = x[x/I\ I] \rightarrow_{\mathtt{dB}\bullet} x[x/y[y/I]] = u$ whose projection is $\mathtt{s}(t) = I\ I \rightarrow_{\mathtt{h}} I = \mathtt{s}(u)$. However, this mismatch arises only if reductions take place inside explicit substitutions. Now, a reduction ρ in λ-calculus induces a reduction ρ' in $\lambda_{\mathtt{sub}}$ where every β-step is simulated by a $\rightarrow_{\mathtt{dB}}$ step followed by a $\rightarrow_{\mathtt{s}}$ step, and in ρ' reductions never take place inside substitutions. It is possible to show that the factorization of ρ' gives a reduction τ with the same properties of ρ', whose projection is then a factorized reduction in λ-calculus. But the proof of this fact is left to a longer version of this paper.

We now show how to generalize the factorization theorem for $\lambda_{\mathtt{sub}}$ into a standardization theorem. In terms of Proof-Nets the factorization theorem says that it is always possible to first reduce at level 0 and then inside arguments of applications and substitutions, which in Proof-Nets are wrapped into !-boxes. The head factorization can be iterated, getting standardization *by levels* in the style of Linear Logic Proof-Nets.

▶ **Definition 3.5** (Boxed contexts). Boxed contexts are indexed by an integer $i \in \mathbb{N}$ called the **level** of the boxed context and defined by:

$$B_0 \ ::= \ H \qquad\qquad B_{i+1} \ ::= \ t\ B_i \mid t[x/B_i] \mid B_0[B_{i+1}]$$

The closure with respect to B_0—which may be confusing—is used to assign level 1 to contexts as $(\lambda x.t\ [\cdot])[x/u]$, for instance. Now, define $\overset{i}{\rightarrow}_{\mathtt{s}} := B_i[\mapsto_{\mathtt{s}}]$, $\overset{i}{\rightarrow}_{\mathtt{dB}} := B_i[\mapsto_{\mathtt{dB}}]$ and $\overset{i}{\rightarrow}_{\lambda_{\mathtt{sub}}} := \overset{i}{\rightarrow}_{\mathtt{s}} \cup \overset{i}{\rightarrow}_{\mathtt{dB}}$. Moreover, define $\overset{>i}{\rightarrow}_{\mathtt{s}} := \cup_{j>i} \overset{j}{\rightarrow}_{\mathtt{s}}$, and similarly for $\overset{>i}{\rightarrow}_{\mathtt{dB}}$ and $\overset{>i}{\rightarrow}_{\lambda_{\mathtt{sub}}}$ (and for $\geq$). According to our definitions we have that $\rightarrow_{\lambda_{\mathtt{sub}}\circ} = \overset{0}{\rightarrow}_{\lambda_{\mathtt{sub}}}$ and $\rightarrow_{\lambda_{\mathtt{sub}}\bullet} = \overset{>0}{\rightarrow}_{\lambda_{\mathtt{sub}}}$. Lemma 3.2 can be generalized as follows:

▶ **Lemma 3.6** (Factorization by levels). *The following swaps hold:*

1. $\overset{>i}{\rightarrow}_{\mathtt{dB}} \overset{i}{\rightarrow}_{\mathtt{dB}} \subseteq \overset{i}{\rightarrow}_{\mathtt{dB}} \overset{>i}{\rightarrow}_{\mathtt{dB}}$.
2. $\overset{>i}{\rightarrow}_{\mathtt{s}} \overset{i}{\rightarrow}_{\mathtt{s}} \subseteq \overset{i}{\rightarrow}_{\mathtt{s}}^{+} \overset{>i}{\rightarrow}_{\mathtt{s}}^{*}$.
3. $\overset{>i}{\rightarrow}_{\mathtt{s}} \overset{i}{\rightarrow}_{\mathtt{dB}} \subseteq \overset{i}{\rightarrow}_{\mathtt{dB}} \overset{>i}{\rightarrow}_{\mathtt{s}}$.
4. $\overset{>i}{\rightarrow}_{\mathtt{dB}} \overset{i}{\rightarrow}_{\mathtt{s}} \subseteq \overset{i}{\rightarrow}_{\mathtt{s}}^{+} \overset{>i}{\rightarrow}_{\mathtt{dB}}^{*}$.

Thus $(\overset{>i}{\rightarrow}_{\mathtt{dB}}, \overset{i}{\rightarrow}_{\mathtt{dB}}, \overset{>i}{\rightarrow}_{\mathtt{s}}, \overset{i}{\rightarrow}_{\mathtt{s}})$ *is a SFS and* $\overset{\geq i}{\rightarrow}_{\lambda_{\mathtt{sub}}} = \overset{i}{\rightarrow}_{\lambda_{\mathtt{sub}}}^{*} \overset{>i}{\rightarrow}_{\lambda_{\mathtt{sub}}}^{*}$.

Proof. We prove the first point, the others are proved analogously. If $t \overset{j}{\rightarrow}_{\mathtt{dB}} u \overset{i}{\rightarrow}_{\mathtt{dB}} v$ with $j > i$ then there are two cases. 1) there exists a context B_i s.t. $t = B_i[w]$, $u = B_i[w']$ and $v = B_i[w'']$ with $w \overset{i-j}{\rightarrow}_{\mathtt{dB}} w' \overset{0}{\rightarrow}_{\mathtt{dB}} w''$, which is equivalent to $w \rightarrow_{\mathtt{dB}\bullet} w' \rightarrow_{\mathtt{dB}\circ} w''$. By Lemma 3.2.1 we get $w \rightarrow_{\mathtt{dB}\circ} \rightarrow_{\mathtt{dB}\bullet} w''$, *i.e.* $w \overset{0}{\rightarrow}_{\mathtt{dB}} \overset{>0}{\rightarrow}_{\mathtt{dB}} w''$, and so $t \overset{i}{\rightarrow}_{\mathtt{dB}} \overset{>i}{\rightarrow}_{\mathtt{dB}} v$. 2) There is no such B_i, then the two steps are independent and permute. ◀

As a corollary we get the standardization theorem by levels for Linear Logic Proof-Nets [15]. Define the level k of a term u as the maximum k s.t. $u = B_k[v]$ for some v.

▶ **Corollary 3.7** (standardization by levels). *If* $t \rightarrow_{\lambda_{\mathtt{sub}}}^{*} u$ *and* k *is the level of* u *then* $t \overset{0}{\rightarrow}_{\lambda_{\mathtt{sub}}}^{*} \overset{1}{\rightarrow}_{\lambda_{\mathtt{sub}}}^{*} \ldots \overset{k-1}{\rightarrow}_{\lambda_{\mathtt{sub}}}^{*} \overset{k}{\rightarrow}_{\lambda_{\mathtt{sub}}}^{*} u$.

Proof. By definition $\to^*_{\lambda_{\mathsf{sub}}} = \overset{\geq 0^*}{\Rightarrow}_{\lambda_{\mathsf{sub}}}$. By lemma 3.6 there exists u_0 s.t. $t \overset{0}{\to}^*_{\lambda_{\mathsf{sub}}} u_0 \overset{\geq 0}{\to}^*_{\lambda_{\mathsf{sub}}} u$. Then we iterate on $u_0 \overset{\geq 0}{\to}^*_{\lambda_{\mathsf{sub}}} u$, getting that there exists u_1 s.t. $u_0 \overset{1}{\to}^*_{\lambda_{\mathsf{sub}}} u_1 \overset{\geq 1}{\to}^*_{\lambda_{\mathsf{sub}}} u$, and so on, until we get $t \overset{0}{\to}^*_{\lambda_{\mathsf{sub}}} u_0 \overset{1}{\to}^*_{\lambda_{\mathsf{sub}}} u_1 \ldots \overset{k-1}{\to}^*_{\lambda_{\mathsf{sub}}} u_{k-1} \overset{k}{\to}^*_{\lambda_{\mathsf{sub}}} u_k \overset{>k}{\to}^*_{\lambda_{\mathsf{sub}}} u$. Now, consider $u_k \overset{>k}{\to}^*_{\lambda_{\mathsf{sub}}} u$. This reduction is empty, otherwise u would have level greater than k, which is absurd. ◀

The paper continues as follows: next section introduces some abstract lemmas for postponements of reductions; Section 5 proves the factorization theorem for SFSs in an abstract way. Sections 6 and 7 give some more applications of the factorization theorem.

4 An abstract toolbox for postponement

In this section we prove four abstract lemmas about permutation properties, which are used to prove the abstract factorization theorem in Section 5. Let S be a rewriting system having three reductions $\to_1$, $\to_2$, and $\to_3$, and for $x, y \in \{1,2,3\}$ note $\to_{x,y} = \to_x \cup \to_y$.

Let us recall some rewriting notions from [31]. What we call factorization is a form of postponement: $\to_2$ **postpones** after $\to_1$ if $\to^*_{1,2} \subseteq \to^*_1 \to^*_2$. Postponement is equivalent to commutation. Two reductions $\to_1$ and $\to_2$ **commute** iff:

$$\begin{tikzcd} t \arrow[r, "*1"] \arrow[d, "*2"'] & u & & & t \arrow[r, "*1"] \arrow[d, "*2"'] & u \arrow[d, "*2"] \\ v & & \text{implies } \exists w \text{ s.t.} & & v \arrow[r, "*1"] & w \end{tikzcd}$$

It is easy to show that $\to_2$ postpones after $\to_1$ iff $\to_1$ commutes with ${}_2\!\leftarrow$. Therefore, lemmas about commutations can be turned into lemmas about postponement, and conversely. Commutation/postponement is a generalization of confluence, indeed a reduction $\to$ is confluent iff it commutes with itself (iff $\leftarrow$ postpones after $\to$).

The first lemma proves postponement from a semi-local hypothesis, *i.e.* from a swap of $\to^*_2$ and $\to_1$ (and not of $\to_2$ and $\to_1$). Afterwards, we show two different local conditions for the semi-local hypothesis. This and the lemma which will follow are standard, we prove them explicitely only to help the unacquainted reader.

▶ **Lemma 4.1** (Semi-local condition for postponement). *If* $\to^*_2 \to_1 \subseteq \to^*_1 \to^*_2$ *then* $\to^*_{1,2} \subseteq \to^*_1 \to^*_2$.

Proof. By induction on the number k of $\to_1$ steps in $\tau : t \to^*_{1,2} u$. The case $k = 0$ is trivial. Let $k > 0$. Then if τ is not of the form of the statement it has the following form: $t \to^*_1 \to^+_2 \to_1 \to^*_{1,2} u$. Using the hypothesis we get: $t \to^*_1 \to^*_1 \to^*_2 \to^*_{1,2} u$. The *i.h.* on $\to^*_2 \to^*_{1,2}$ gives: $t \to^*_1 \to^*_1 \to^*_1 \to^*_2 u$ and we conclude. ◀

The first local condition is that $\to_2 \to_1 \subseteq \to_1 \to^*_2$. Note that one step of $\to_1$ on the left becomes one step of $\to_1$ on the right. The right part can also be replaced by $\to_1 \to^*_2 \cup \to^*_2$, but not by $\to^*_1 \to^*_2$, otherwise the inductive argument does not work and it is not difficult to build a counter-example (see [9]).

▶ **Lemma 4.2** (Local condition for postponement). *If* $\to_2 \to_1 \subseteq \to_1 \to^*_2$ *then* $\to^*_2 \to_1 \subseteq \to_1 \to^*_2$ *and so* $\to^*_{1,2} \subseteq \to^*_1 \to^*_2$.

Proof. We prove the first consequence, the second one follows from the first and Lemma 4.1. By induction on k, where $t \to^k_2 \to_1 u$. The case $k = 0$ is trivial. Let $k > 0$. Then $t \to^{k-1}_2 \to_2 \to_1 u$ and applying the hypothesis to the suffix we get $t \to^{k-1}_2 \to_1 \to^*_2 u$. We conclude applying the *i.h.* to the prefix $\to^{k-1}_2 \to_1$. ◀

In order to get a factorization theorem for SFSs we have to deal with permutations of the form $\to_2\to_1\subseteq\to_1\to_1\to_2^*$, given for instance by Lemma 3.2.2. These commutations are harmless if $\to_1$ is strongly normalizing, as next lemma shows. The lemma has been independently proved by Geser in his Ph.D. thesis [17] and by Di Cosmo and Piperno in [10], where it is presented for commutation, and it is here adapted to postponement. It can be seen as a generalization of Newman's lemma.

▶ **Lemma 4.3** (Geser-Di Cosmo-Piperno's lemma, adapted). *If $\to_2\to_1\subseteq\to_1^+\to_2^*$ and $\to_1$ is strongly normalizing then $\to_2^*\to_1^+\subseteq\to_1^+\to_2^*$, and so $\to_{1,2}^*\subseteq\to_1^*\to_2^*$.*

Proof. We prove the first consequence, the second one follows from the first and Lemma 4.1. Let $\tau : t \to_2^k\to_1^h u$. By induction[3] on the pair $(\eta(t), k)$, where $\eta(t)$ is the length of the maximal $\to_1$ reduction from t, ordered lexicographically. The case $k = 0$ is trivial, then let $k > 0$. Now, if $\tau =\to_2^{k-1}\to_2\to_1\to_1^{h-1}$ then by applying the first hypothesis to the central subesequence we get $\tau \subseteq\to_2^{k-1}\to_1^+\to_2^*\to_1^{h-1}$. The measure of the prefix $\to_2^{k-1}\to_1^+$ is $(\eta(t), k-1)$ hence by *i.h.* we get $\tau \subseteq\to_1^+\to_2^*\to_2^*\to_1^{h-1}$. For the suffix $\to_2^*\to_2^*\to_1^{h-1}$ it is the first component of the measure which decreases, since its starting term is obtained through a $\to_1^+$-reduction from t, so one can apply the *i.h.* and get $\tau \subseteq\to_1^+\to_1^+\to_2^*$. ◀

The last abstract lemma we need is the one for the diagonal swaps, which has to take into account the phenomenon of a third reduction popping out in the right part, *i.e.* permutations of the form $\to_2\to_1\subseteq\to_1\to_{2,3}^*$ (given for instance by Lemma 3.2.4). We prove a sort of generalization of Lemma 4.2 to this setting. This lemma is the key for the abstract factorization theorem of next section. It is an original contribution of the paper.

▶ **Lemma 4.4.** *If*
1. $\to_2\to_1\subseteq\to_1\to_{2,3}^*$ *and*
2. $\to_2\to_3\subseteq\to_3^+\to_2^*$ *and*
3. $\to_3$ *is strongly normalizing*

then $\to_2^*\to_1^*\subseteq\to_{1,3}^*\to_2^*$.

Proof. First of all note that the second and third hypothesis allow to apply Lemma 4.3 and get $\to_{2,3}^*\subseteq\to_3^*\to_2^*$, let us call this fact property (#). The proof is by induction on (h, k), where $\to_2^k\to_1^h$, ordered lexicographically. If $k = 0$ or $h = 0$ it is trivial. If $k > 0$ and $h > 0$ then applying the first hypothesis to the central part of $\tau :\to_2^{k-1}\to_2\to_1\to_1^{h-1}$ we get:

$$\tau \subseteq\to_2^{k-1}\to_1\to_{2,3}^*\to_1^{h-1}\subseteq_{(\#)}\to_2^{k-1}\to_1\to_3^*\to_2^*\to_1^{h-1}$$

Then by applying the *i.h.* to the prefix $\to_2^{k-1}\to_1$ we get:

$$\tau \subseteq\to_{1,3}^*\underbrace{\to_2^*\to_3^*\to_2^*}_{\to_{2,3}^*}\to_1^{h-1}\subseteq_{(\#)}\underbrace{\to_{1,3}^*\to_3^*}_{\to_{1,3}^*}\to_2^*\to_1^{h-1}\subseteq\to_{1,3}^*\to_2^*\to_1^{h-1}$$

Now by applying the *i.h.* to the suffix $\to_2^*\to_1^{h-1}$ we conclude with $\tau \subseteq \underbrace{\to_{1,3}^*\to_{1,3}^*}_{\to_{1,3}^*}\to_2^*\subseteq\to_{1,3}^*\to_2^*$

◀

[3] For the sake of simplicity the proof implicitly assumes that S is a finitely branching rewriting system. This assumption can be dropped by replacing the induction on $\eta(t)$ with an induction on the well-founded order on the elements of S induced by $\to_1$ (because of strong normalization). We learned this from Roberto Di Cosmo.

5 Abstract Standardization

In this section we prove that any square factorization system enjoys a factorization theorem. Let us recall the definition from Section 3.

▶ Definition. A **square factorization system** (SFS) S is given by a set $|S|$ and four reduction relations $(\rightsquigarrow_\bullet, \rightsquigarrow_\circ, \rightarrowtail_\bullet, \rightarrowtail_\circ)$ on $|S|$ s. t. the following conditions hold (remember that $\rightsquigarrow := \rightsquigarrow_\circ \cup \rightsquigarrow_\bullet$, $\rightarrowtail := \rightarrowtail_\circ \cup \rightarrowtail_\bullet$, $\to_\circ := \rightsquigarrow_\circ \cup \rightarrowtail_\circ$, $\to_\bullet := \rightsquigarrow_\bullet \cup \rightarrowtail_\bullet$ and $\to := \rightsquigarrow \cup \rightarrowtail$):

1. **Termination**: $\rightsquigarrow_\circ$ and $\rightarrowtail_\circ$ are strongly normalizing.
2. **Row-swap 1**: $\rightsquigarrow_\bullet\rightsquigarrow_\circ \subseteq \rightsquigarrow_\circ^+\rightsquigarrow_\bullet^*$.
3. **Row-swap 2**: $\rightarrowtail_\bullet\rightarrowtail_\circ \subseteq \rightarrowtail_\circ^+\rightarrowtail_\bullet^*$.
4. **Diagonal-swap 1**: $\rightarrowtail_\bullet\rightsquigarrow_\circ \subseteq \rightsquigarrow_\circ\rightarrowtail^*$.
5. **Diagonal-swap 2**: $\rightsquigarrow_\bullet\rightarrowtail_\circ \subseteq \rightarrowtail_\circ\rightsquigarrow^*$.

The proof relies on the abstract lemmas of the previous sections and is simple. We just lift the local permutations demanded to a SFS to sequences of steps (Points 1.a, 1.b, 2.a, 2.b below) and then glue them together (Points 1.c and 2.c).

▶ **Lemma 5.1** (Induced permutations of $\bullet^*$ and $\circ^*$). *Let $S = \{\rightsquigarrow_\bullet, \rightsquigarrow_\circ, \rightarrowtail_\bullet \rightarrowtail_\circ\}$ be a square factorization system. The following inclusions hold:*

1) **Swap of $\rightarrowtail_\bullet^*$ and $\to_\circ^*$***:*
- *a)* ***Diagonal:*** $\rightarrowtail_\bullet^*\rightsquigarrow_\circ^*\subseteq\to_\circ^*\rightarrowtail_\bullet^*$.
- *b)* ***Row:*** $\rightarrowtail_\bullet^*\rightarrowtail_\circ^*\subseteq\rightarrowtail_\circ^*\rightarrowtail_\bullet^*$.
- *c)* ***Diagonal + Row:*** $\rightarrowtail_\bullet^*\to_\circ^*\subseteq\to_\circ^*\rightarrowtail_\bullet^*$.

2) **Swap of $\rightsquigarrow_\bullet^*$ and $\to_\circ^*$***:*
- *a)* ***Diagonal:*** $\rightsquigarrow_\bullet^*\rightarrowtail_\circ^*\subseteq\to_\circ^*\rightsquigarrow_\bullet^*$.
- *b)* ***Row:*** $\rightsquigarrow_\bullet^*\rightsquigarrow_\circ^*\subseteq\rightsquigarrow_\circ^*\rightsquigarrow_\bullet^*$.
- *c)* ***Diagonal + Row:*** $\rightsquigarrow_\bullet^*\to_\circ^*\subseteq\to_\circ^*\rightsquigarrow_\bullet^*$.

Proof. 1.a) Conditions 4 ($\rightarrowtail_\bullet\rightsquigarrow_\circ\subseteq\rightsquigarrow_\circ\rightarrowtail^*$), 3 ($\rightarrowtail_\bullet\rightarrowtail_\circ\subseteq\rightarrowtail_\circ^+\rightarrowtail_\bullet^*$) and 1 ($\rightsquigarrow_\circ$ is strongly normalizing) for SFS satisfy the hypothesis of Lemma 4.4 (with $\to_1=\rightsquigarrow_\circ$, $\to_2=\rightarrowtail_\bullet$, $\to_3=\rightarrowtail_\circ$, $\to_{2,3}=(\rightarrowtail_\bullet\cup\rightarrowtail_\circ)=\rightarrowtail$ and $\to_{1,3}=(\rightarrowtail_\circ\cup\rightsquigarrow_\circ)=\to_\circ$). Hence, we get that $\rightarrowtail_\bullet^*\rightsquigarrow_\circ^*$ is included in $\to_\circ^*\rightarrowtail_\bullet^*$.
1.b) Conditions 1 ($\rightarrowtail_\circ$ is strongly normalizing) and 3 ($\rightarrowtail_\bullet\rightarrowtail_\circ\subseteq\rightarrowtail_\circ^+\rightarrowtail_\bullet^*$) for SFS satisfy the hypothesis of Lemma 4.3 (with $\to_1=\rightarrowtail_\circ$, $\to_2=\rightarrowtail_\bullet$). Hence, we get $(\rightarrowtail_\circ\cup\rightarrowtail_\bullet)^*\subseteq\rightarrowtail_\circ^*\rightarrowtail_\bullet^*$. We conclude, since $\rightarrowtail_\bullet^*\rightarrowtail_\circ^*$ is included in $(\rightarrowtail_\circ\cup\rightarrowtail_\bullet)^*$ and thus in $\rightarrowtail_\circ^*\rightarrowtail_\bullet^*$.
1.c) From 1.a and 1.b we get $\rightarrowtail_\bullet^*\rightsquigarrow_\circ\subseteq\to_\circ^*\rightarrowtail_\bullet^*$ and $\rightarrowtail_\bullet^*\rightarrowtail_\circ\subseteq\rightarrowtail_\circ^*\rightarrowtail_\bullet^*$, hence $\rightarrowtail_\bullet^*\to_\circ\subseteq\to_\circ^*\rightarrowtail_\bullet^*$. By Lemma 4.1 we get $(\to_\circ\cup\rightarrowtail_\bullet)^*\subseteq\to_\circ^*\rightarrowtail_\bullet^*$ and we conclude, since $\rightarrowtail_\bullet^*\to_\circ^*\subseteq(\to_\circ\cup\rightarrowtail_\bullet)^*$.
2.a) Conditions 5 ($\rightsquigarrow_\bullet\rightarrowtail_\circ\subseteq\rightarrowtail_\circ\rightsquigarrow^*$), 2 ($\rightsquigarrow_\bullet\rightsquigarrow_\circ\subseteq\rightsquigarrow_\circ^+\rightsquigarrow_\bullet^*$) and 1 ($\rightarrowtail_\circ$ is strongly normalizing) for SFS satisfy the hypothesis of Lemma 4.4 (with $\to_1=\rightarrowtail_\circ$, $\to_2=\rightsquigarrow_\bullet$, $\to_3=\rightsquigarrow_\circ$, $\to_{2,3}=(\rightsquigarrow_\bullet\cup\rightsquigarrow_\circ)=\rightsquigarrow$ and $\to_{1,3}=(\rightarrowtail_\circ\cup\rightsquigarrow_\circ)=\to_\circ$). Hence, we get that $\rightsquigarrow_\bullet^*\rightarrowtail_\circ^*$ is included in $\to_\circ^*\rightsquigarrow_\bullet^*$.
2.b) Conditions 1 ($\rightsquigarrow_\circ$ is strongly normalizing) and 2 ($\rightsquigarrow_\bullet\rightsquigarrow_\circ\subseteq\rightsquigarrow_\circ^+\rightsquigarrow_\bullet^*$) for SFS satisfy the hypothesis of Lemma 4.3 (with $\to_1=\rightsquigarrow_\circ$, $\to_2=\rightsquigarrow_\bullet$). Hence, we get $(\rightsquigarrow_\circ\cup\rightsquigarrow_\bullet)^*\subseteq\rightsquigarrow_\circ^*\rightsquigarrow_\bullet^*$. We conclude, since $\rightsquigarrow_\bullet^*\rightsquigarrow_\circ^*$ is included in $(\rightsquigarrow_\circ\cup\rightsquigarrow_\bullet)^*$ and thus in $\rightsquigarrow_\circ^*\rightsquigarrow_\bullet^*$.
2.c) From 2.a and 2.b we get $\rightsquigarrow_\bullet^*\rightarrowtail_\circ\subseteq\to_\circ^*\rightsquigarrow_\bullet^*$ and $\rightsquigarrow_\bullet^*\rightsquigarrow_\circ\subseteq\rightsquigarrow_\circ^*\rightsquigarrow_\bullet^*$, hence $\rightsquigarrow_\bullet^*\to_\circ\subseteq\to_\circ^*\rightsquigarrow_\bullet^*$. By Lemma 4.1 we get $(\to_\circ\cup\rightsquigarrow_\bullet)^*\subseteq\to_\circ^*\rightsquigarrow_\bullet^*$ and we conclude, since $\rightsquigarrow_\bullet^*\to_\circ^*\subseteq(\to_\circ\cup\rightsquigarrow_\bullet)^*$. ◀

It is now easy to conclude.

▶ **Theorem 5.2.** *Let $S = \{\rightsquigarrow_\bullet, \rightsquigarrow_\circ, \rightarrowtail_\bullet \rightarrowtail_\circ\}$ be a square factorization system. Then:*

1. **Swap of $\bullet$ and $\circ^*$***:* $\to_\bullet\to_\circ^*\subseteq\to_\circ^*\to_\bullet^*$.
2. **Factorization***:* $\to^*\subseteq\to_\circ^*\to_\bullet^*$.

Proof. **1.** From points 1.c and 2.c of Lemma 5.1 we get $\rightarrowtail_\bullet\to_\circ^*\subseteq\to_\circ^*\rightarrowtail_\bullet^*$ and $\rightsquigarrow_\bullet\to_\circ^*\subseteq\to_\circ^*\rightsquigarrow_\bullet^*$, hence $\to_\bullet\to_\circ^*\subseteq\to_\circ^*\to_\bullet^*$.

2. By induction on the length k of $\to^*$. The case $k = 0$ is trivial, then let $k > 0$. By *i.h.* we get $\to\to^{k-1}\subseteq\to\to_\circ^*\to_\bullet^*$. If the first step is a $\to_\circ$ step then there is nothing to prove. Otherwise, by the previous point we get $\to_\bullet\to_\circ^*\to_\bullet^*\subseteq\to_\circ^*\to_\bullet^*\to_\bullet^*$ and conclude.

◀

All SFSs in the paper are composed of four rules. But some substitution calculi have many rules, in particular a set of rules for substitutions. To apply the theorem is enough to identify this set with $\rightarrowtail$, and to split each rule accordingly. Similarly, there can also be various rules for creating substitutions, whose union would be $\rightsquigarrow$. The axioms for a SFS are symmetric, so the choice between $\rightarrowtail$ or $\rightsquigarrow$ for one or the other set does not really matter.

6 The value-substitution calculus $\lambda_{\mathtt{vsub}}$

The value-substitution calculus $\lambda_{\mathtt{vsub}}$ has the same syntax as $\lambda_{\mathtt{sub}}$, but we distinguish the syntactic category of values:

$$\mathsf{v} = x \mid \lambda x.t$$

The rewriting relation $\to_{\lambda_{\mathtt{vsub}}}$ is defined as the union of $\to_{\mathtt{dB}}$ (defined as before) and $\to_{\mathtt{vs}}$, which is the context closure of:

$$t[x/\mathsf{v}\mathtt{L}] \quad \mapsto_{\mathtt{vs}} \quad t\{x/\mathsf{v}\}\mathtt{L}$$

Remark rule $\mapsto_{\mathtt{vs}}$: there is no typo, in the left-hand side $\mathtt{L}$ is inside $[x/\mathsf{v}\mathtt{L}]$ and in the right-hand side it is outside $\{x/\mathsf{v}\}$. A detailed account of the value-substitution calculus, which is confluent, can be found in [8]. For $\lambda_{\mathtt{vsub}}$ we are going to prove two different factorization theorems, both based on the following property.

▶ **Lemma 6.1** ([8]). *$\to_{\mathtt{dB}}$ and $\to_{\mathtt{vs}}$ are strongly normalizing and confluent.*

The first factorization scheme for $\to_{\lambda_{\mathtt{vsub}}}$ is given by weak reduction (*i.e.* reductions out of lambdas) and non-weak reductions, playing for $\lambda_{\mathtt{vsub}}$ the role of head and internal reductions for $\lambda_{\mathtt{sub}}$. **Weak contexts** are contexts whose hole is not under an abstraction:

$$W ::= [\cdot] \mid W\ t \mid t\ W \mid W[x/t] \mid t[x/W]$$

Weak reduction $\rightharpoonup_\circ$ is defined as the union of $\rightharpoonup_{\mathtt{dB}\circ}$ and $\rightharpoonup_{\mathtt{vs}\circ}$, which are obtained as the closure by *weak* contexts of $\mapsto_{\mathtt{dB}}$ and $\mapsto_{\mathtt{vs}}$. Note that the reductions $\rightharpoonup_{\mathtt{vs}\circ}$ and $\rightharpoonup_{\mathtt{dB}\circ}$ are strongly normalizing because their unrestricted versions are. Weak reduction $\rightharpoonup_\circ$ has the diamond property [8].

In order to obtain a square factorization system we need to define the complements of $\rightharpoonup_\circ,\rightharpoonup_{\mathtt{dB}\circ}$ and $\rightharpoonup_{\mathtt{vs}\circ}$. We define $\rightharpoonup_\bullet$ as the union of $\rightharpoonup_{\mathtt{dB}\bullet}:=\to_{\mathtt{dB}} \setminus \rightharpoonup_{\mathtt{dB}\circ}$ and $\rightharpoonup_{\mathtt{vs}\bullet}:=\to_{\mathtt{vs}} \setminus \rightharpoonup_{\mathtt{vs}\circ}$, which are easily seen to be also definable via **non-weak contexts**, given by $\overline{W} ::= \lambda x.C \mid W[\overline{W}]$. Two examples:

$$(\lambda x.t)[x/\mathsf{v}[y/u]]\ s \rightharpoonup_{\mathtt{vs}\circ} (\lambda x.t)\{x/\mathsf{v}\}[y/u]\ s \qquad\qquad t\ \lambda x.(s[x/\mathsf{v}]) \rightharpoonup_{\mathtt{vs}\bullet} t\ \lambda x.(s\{x/\mathsf{v}\})$$

The following theorem is proved by means of a substitution lemma and an induction on the rewriting relations (details in [2]).

▶ **Theorem 6.2** (weak factorization). *The following inclusions hold:*

1. ***Row 1:*** $\rightharpoonup_{\mathtt{dB}\bullet}\rightharpoonup_{\mathtt{dB}\circ}\subseteq\rightharpoonup_{\mathtt{dB}\circ}\rightharpoonup_{\mathtt{dB}\bullet} \cup \rightharpoonup_{\mathtt{dB}\circ}\rightharpoonup_{\mathtt{dB}\circ}$.
2. ***Row 2:*** $\rightharpoonup_{\mathtt{vs}\bullet}\rightharpoonup_{\mathtt{vs}\circ}\subseteq\rightharpoonup_{\mathtt{vs}\circ}\rightharpoonup_{\mathtt{vs}\bullet}^*$.

3. ***Diagonal 1:*** $\rightharpoonup_{\mathtt{vs}\bullet}\rightharpoonup_{\mathtt{dB}\circ}\subseteq\rightharpoonup_{\mathtt{dB}\circ}\rightharpoonup_{\mathtt{vs}\circ}\cup\rightharpoonup_{\mathtt{dB}\circ}\rightharpoonup_{\mathtt{vs}\bullet}$.
4. ***Diagonal 2:*** $\rightharpoonup_{\mathtt{dB}\bullet}\rightharpoonup_{\mathtt{vs}\circ}\subseteq\rightharpoonup_{\mathtt{vs}\circ}\rightharpoonup^{*}_{\mathtt{dB}\bullet}$.

Thus $(\rightharpoonup_{\mathtt{dB}\bullet},\rightharpoonup_{\mathtt{dB}\circ},\rightharpoonup_{\mathtt{vs}\bullet},\rightharpoonup_{\mathtt{vs}\circ})$ *is a SFS and* $\to^{*}_{\lambda_{\mathtt{vsub}}}=\rightharpoonup^{*}_{\circ}\rightharpoonup^{*}_{\bullet}$.

We invite the reader to compare the diagrams of $\lambda_{\mathtt{sub}}$ (Lemma 3.2) and $\lambda_{\mathtt{vsub}}$ (Theorem 6.2): they are similar and yet different, despite both are SFS.

At this point we could iterate the factorization as for $\lambda_{\mathtt{sub}}$, but we prefer to show another factorization scheme obtained by combining head reduction and weak reduction in a new interesting way. The following grammar defines stratified-weak contexts SW:

$$SW ::= W \mid SW\ t \mid \lambda x.SW \mid SW[t/x]$$

Note that equivalently $SW ::= H[W]$. Stratified-weak reduction $\rightharpoondown_{\circ}$ is defined as the union of $\rightharpoondown_{\mathtt{dB}\circ}$ and $\rightharpoondown_{\mathtt{vs}\circ}$ the closures by stratified-weak contexts $\mapsto_{\mathtt{dB}}$ and $\mapsto_{\mathtt{vs}}$. Stratified-weak reduction $\rightharpoondown_{\circ}$ has the diamond property [8]. We also need the complements of $\rightharpoondown_{\circ}$,$\rightharpoondown_{\mathtt{dB}\circ}$ and $\rightharpoondown_{\mathtt{vs}\circ}$. Let us set: $\rightharpoondown_{\bullet}:=\to\setminus\rightharpoondown_{\circ}$, $\rightharpoondown_{\mathtt{dB}\bullet}:=\to_{\mathtt{dB}}\setminus\rightharpoondown_{\mathtt{dB}\circ}$ and $\rightharpoondown_{\mathtt{vs}\bullet}:=\to_{\mathtt{vs}}\setminus\rightharpoondown_{\mathtt{vs}\circ}$. Two examples:

$$\lambda x.(s[x/\mathsf{v}])\rightharpoondown_{\mathtt{vs}\circ}\lambda x.(s\{x/\mathsf{v}\}) \qquad t[y/\lambda x.(s[x/\mathsf{v}])]\rightharpoondown_{\mathtt{vs}\bullet}t[y/\lambda x.(s\{x/\mathsf{v}\})]$$

The following theorem is proved by means of a substitution lemma and by induction on the rewriting relations (details in [2]).

▶ **Theorem 6.3** (stratified-weak factorization). *The following inclusions hold:*

1. ***Row 1:*** $\rightharpoondown_{\mathtt{dB}\bullet}\rightharpoondown_{\mathtt{dB}\circ}\subseteq\rightharpoondown_{\mathtt{dB}\circ}\rightharpoondown_{\mathtt{dB}\bullet}\cup\rightharpoondown_{\mathtt{dB}\circ}\rightharpoondown_{\mathtt{dB}\circ}$.
2. ***Row 2:*** $\rightharpoondown_{\mathtt{vs}\bullet}\rightharpoondown_{\mathtt{vs}\circ}\subseteq\rightharpoondown_{\mathtt{vs}\circ}\rightharpoondown^{*}_{\mathtt{vs}\bullet}\cup\rightharpoondown_{\mathtt{vs}\circ}\rightharpoondown_{\mathtt{vs}\circ}\rightharpoondown^{*}_{\mathtt{vs}\bullet}$.
3. ***Diagonal 1:*** $\rightharpoondown_{\mathtt{vs}\bullet}\rightharpoondown_{\mathtt{dB}\circ}\subseteq\rightharpoondown_{\mathtt{dB}\circ}\rightharpoondown_{\mathtt{vs}\bullet}\cup\rightharpoondown_{\mathtt{dB}\circ}\rightharpoondown_{\mathtt{vs}\circ}$.
4. ***Diagonal 2:*** $\rightharpoondown_{\mathtt{dB}\bullet}\rightharpoondown_{\mathtt{vs}\circ}\subseteq\rightharpoondown_{\mathtt{vs}\circ}\rightharpoondown_{\mathtt{dB}\circ}\rightharpoondown^{*}_{\mathtt{dB}\bullet}\cup\rightharpoondown_{\mathtt{vs}\circ}\rightharpoondown^{*}_{\mathtt{dB}\bullet}$.

Thus $(\rightharpoondown_{\mathtt{dB}\bullet},\rightharpoondown_{\mathtt{dB}\circ},\rightharpoondown_{\mathtt{vs}\bullet},\rightharpoondown_{\mathtt{vs}\circ})$ *is a SFS and* $\to^{*}_{\lambda_{\mathtt{vsub}}}=\rightharpoondown^{*}_{\circ}\rightharpoondown^{*}_{\bullet}$.

Note that once again the permutations are different from the previous cases, and yet caught by the axioms for a SFS. The two factorization theorems for $\lambda_{\mathtt{vsub}}$ have been used in [8] to give a new operational characterization of call-by-value solvability.

7 The linear substitution calculus $\lambda_{\mathtt{lsub}}$

In this section we apply the factorization theorem to a finer calculus of explicit substitutions, the linear substitution calculus $\lambda_{\mathtt{lsub}}$, a slight variation over a calculus introduced by Robin Milner in [29], which can also be seen as a simplification of the structural λ-calculus $\lambda\mathtt{j}$ [5]. The linear substitution calculus has the the same syntax as $\lambda_{\mathtt{sub}}$. The rewriting rules of $\lambda_{\mathtt{lsub}}$ are $\to_{\mathtt{dB}}$ (defined as before), $\to_{\mathtt{ls}}$ and $\to_{\mathtt{gc}}$, which are defined as the context closures of:

$$C[x][x/u] \quad \mapsto_{\mathtt{ls}} \quad C[u][x/u] \qquad\qquad t[x/u] \quad \mapsto_{\mathtt{gc}} \quad t \qquad \text{if } x\notin\mathtt{fv}(t)$$

Rule $\mapsto_{\mathtt{ls}}$ uses contexts: whenever we write $C[x][x/u]$ we implicitly assume that C does not capture x, despite the fact that in general (for instance in the context closure of the rules) contexts may capture variables. We also note $\to_{\lambda_{\mathtt{lsub}}}=\to_{\mathtt{dB}}\cup\to_{\mathtt{ls}}\cup\to_{\mathtt{gc}}$. Note that $\lambda_{\mathtt{lsub}}$ is obtained from $\lambda_{\mathtt{sub}}$ by decomposing the substitution rule in two more atomic rules. The difference between $\lambda_{\mathtt{lsub}}$ and the original calculus of Milner [29] is rule $\to_{\mathtt{dB}}$, which is not at a distance in Milner's calculus (*i.e.* it has the form of the $\mathtt{B}$-rule in the introducion).

The linear substitution calculus enjoys all properties demanded to explicit substitution calculi, obtained by easy adaptations of the proofs for $\lambda\mathtt{j}$ in [5]. In particular, it is confluent and preserves β-strong normalization. We will use the following property:

▶ **Lemma 7.1.** $\to_{\mathtt{dB}}$, $\to_{\mathtt{ls}}$, $\to_{\mathtt{gc}}$ *are strongly normalizing and confluent.*

Another important property is that garbage collection can be postponed.

▶ **Lemma 7.2.** *Let* $\to_{\neg\mathtt{gc}}$ *be* $\to_{\mathtt{dB}} \cup \to_{\mathtt{ls}}$, i.e. *the complement of* $\to_{\mathtt{gc}}$ *with respect to* $\to_{\lambda_{\mathtt{lsub}}}$.
1. ***Local postponement of garbage collection:*** $\to_{\mathtt{gc}}\to_{\neg\mathtt{gc}}\subseteq\to_{\neg\mathtt{gc}}\to_{\mathtt{gc}}^{*}$.
2. ***Postponement of garbage collection:*** $\to_{\lambda_{\mathtt{lsub}}}^{*}\subseteq\to_{\neg\mathtt{gc}}^{*}\to_{\mathtt{gc}}^{*}$.

Proof. 1) By induction on $\to_{\mathtt{gc}}$. 2) It follows from the previous point and Lemma 4.2. ◀

Interestingly, if one defines head reduction as for $\lambda_{\mathtt{sub}}$, *i.e.* as the closure by head contexts of the root rewriting rules, the factorization theorem does not hold. Consider this reduction:

$$x[x/y[y/z]][z/u] \to_{\mathtt{ls}} x[x/z[y/z]][z/u] \to_{\mathtt{ls}} x[x/u[y/z]][z/u]$$

the first step has box level 1, while the second has box level 0, but they cannot be permuted[4]. As we will see later, the notion of level of the linear substitution calculus is more subtle.

In $\lambda_{\mathtt{lsub}}$ it is possible to reformulate what in the Linear Logic literature is called *linear head reduction* [23], and to prove factorization of $\to_{\neg\mathtt{gc}}$ with respect to it. Note that in a $\to_{\mathtt{ls}}$ step contexts are involved twice: to close the rule and to select the variable occurrence to substitute for. For a head *linear* $\to_{\mathtt{ls}}$ step both contexts have to be head.

▶ **Definition 7.3** (head linear reduction)**. Head linear reduction** $\multimap_{\circ}$ is defined as the union of $\multimap_{\mathtt{dB}\circ}:= H[\mapsto_{\mathtt{dB}}]$ and $\multimap_{\mathtt{ls}\circ}:= H[\hat{\multimap}_{\mathtt{ls}}]$, where $\hat{\multimap}_{\mathtt{ls}}$ is given by:

$$H[x][x/u] \quad \hat{\multimap}_{\mathtt{ls}} \quad H[u][x/u]$$

Internal linear reduction $\multimap_{\bullet}$ is defined as the union of $\multimap_{\mathtt{dB}\bullet}:=\to_{\mathtt{dB}} \setminus \multimap_{\mathtt{dB}\circ}$ and $\multimap_{\mathtt{ls}\bullet}:=\to_{\mathtt{ls}} \setminus \multimap_{\mathtt{ls}\circ}$. Two examples:

$$(y\ y)[y/z] \multimap_{\mathtt{ls}\circ} (z\ y)[y/z] \qquad\qquad (y\ y)[y/z] \multimap_{\mathtt{ls}\bullet} (y\ z)[y/z]$$

With these definitions we get the following theorem, proved at the end of the section:

▶ **Theorem 7.4.** $(\multimap_{\mathtt{dB}\bullet}, \multimap_{\mathtt{dB}\circ}, \multimap_{\mathtt{ls}\bullet}, \multimap_{\mathtt{ls}\circ})$ *is a SFS for* $\to_{\neg\mathtt{gc}}$, *and so* $\to_{\neg\mathtt{gc}}^{*}\subseteq\multimap_{\circ}^{*}\multimap_{\bullet}^{*}$.

To our knowledge this is the first result formally proving factorization with respect to linear head reduction. We recall that head linear reduction can be seen as an abstraction of Krivine's abstract machine [14]. Therefore, when β-reduction is decomposed in more atomic rules, the factorization theorem becomes a method to derive abstract machines.

The relation between head linear reduction and the head reduction of λ-calculus has been studied in depth in [7].

It is possible to generalize head linear reduction $\multimap_{\circ}$ into a family $\{\overset{i}{\multimap}\}_{i\in\mathbb{N}}$ of reductions by levels, in which $\multimap_{\circ}$ corresponds to $\overset{0}{\multimap}$. We still use contexts to define the level of a step, but the notion of level used here differs from the one used for $\lambda_{\mathtt{sub}}$: the index counts the arguments of applications only. Contexts whose hole is inside a substitution are noted H_{∞}. **Leveled contexts** are defined as follows:

$$H_0 ::= H \qquad\qquad H_{i+1} ::= t\ H_i \mid H_0[H_{i+1}] \qquad\qquad H_{\infty} ::= t\ [x/C] \mid H_i[H_{\infty}]$$

[4] A note for Proof-Nets experts: this phenomenon is at first sight surprising, but it is not due to our formalism and we are not the firsts to point it out: working at the same time with small-steps rules (as in $\lambda_{\mathtt{lsub}}$) and freely permutating contraction with !-boxes (which is implicit using terms) breaks standardization by levels also in a traditional Proof-Nets syntax, see Tranquilli's thesis [33], p. 133.

In order to define linear reduction by levels we fix some conventions. We will need to sum indexes, and for ∞ we use the equalities $\infty + i = \infty$ and $\infty + \infty = \infty$. Moreover, we note $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$ and consider $i < \infty$ for all $i \in \mathbb{N}$.

▶ **Definition 7.5.** The **linear reduction** $\overset{i}{\multimap}$ is the union of $\overset{i}{\multimap}_{\mathtt{dB}}$ and $\overset{i}{\multimap}_{\mathtt{ls}}$, defined as follows. For $i \in \mathbb{N}^\infty$ the reduction $\overset{i}{\multimap}_{\mathtt{dB}}$ is the closure by H_i-contexts of $\mapsto_{\mathtt{dB}}$. The case of $\to_{\mathtt{ls}}$ is a bit more complex. Let $\overset{j}{\mapsto}_{\mathtt{ls}}$ be defined by the rule:

$$H_j[x][x/u] \overset{j}{\mapsto}_{\mathtt{ls}} H_j[u][x/u]$$

Then define $\overset{i}{\multimap}_{\mathtt{ls}} := H_l[\overset{j}{\mapsto}_{\mathtt{ls}}]$, where $i = l + j$. Define also $\overset{>i}{\multimap}_{\mathtt{ls}} := \cup_{j>i} \overset{j}{\multimap}_{\mathtt{ls}}$ and similarly for $\overset{>i}{\multimap}_{\mathtt{dB}}$, $\overset{>i}{\multimap}$ and for $\geq$. Some examples of $\overset{i}{\multimap}_{\mathtt{ls}}$ reductions:

$$(y\ y)[y/z] \overset{0}{\multimap}_{\mathtt{ls}} (z\ y)[y/z] \qquad (y\ y)[y/z] \overset{1}{\multimap}_{\mathtt{ls}} (y\ z)[y/z] \qquad y[y/z[z/s]] \overset{\infty}{\multimap}_{\mathtt{ls}} y[y/s[z/s]]$$

$$(x\ (y\ y)[y/z]) \overset{2}{\multimap}_{\mathtt{ls}} (x\ (y\ z)[y/z]) \qquad (x\ (y\ y)[y/z])[z/s] \overset{\infty}{\multimap}_{\mathtt{ls}} (x\ (y\ y)[y/s])[z/s]$$

Note that when $t \overset{i}{\multimap}_{\mathtt{ls}} s$ then exists H_i s.t. $t = H_i[x] \overset{i}{\multimap}_{\mathtt{ls}} H_i[u]$ with respect to some substitution $[x/u]$ in H_i.

The following lemma gives a detailled analysis of leveled diagrams for the linear substitution calculus. It is proved by an analysis of diagrams relying on some technical lemmas on leveled contexts (details in [2]).

▶ **Lemma 7.6.** *Let $i < j < \infty$. Then:*

1) ***Row 1a:*** $\overset{j}{\multimap}_{\mathtt{dB}}\overset{i}{\multimap}_{\mathtt{dB}} \subseteq \overset{i}{\multimap}_{\mathtt{dB}}\overset{j}{\multimap}_{\mathtt{dB}} \cup \overset{i}{\multimap}_{\mathtt{dB}}\overset{\infty}{\multimap}_{\mathtt{dB}}$. *4)* ***Diagonal 1a:*** $\overset{j}{\multimap}_{\mathtt{ls}}\overset{i}{\multimap}_{\mathtt{dB}} \subseteq \overset{i}{\multimap}_{\mathtt{dB}}\overset{j}{\multimap}_{\mathtt{ls}} \cup \overset{i}{\multimap}_{\mathtt{dB}}\overset{\infty}{\multimap}_{\mathtt{ls}}$.

2) ***Row 1b:*** $\overset{\infty}{\multimap}_{\mathtt{dB}}\overset{i}{\multimap}_{\mathtt{dB}} \subseteq \overset{i}{\multimap}_{\mathtt{dB}}\overset{\infty}{\multimap}_{\mathtt{dB}}$. *5)* ***Diagonal 1b:*** $\overset{\infty}{\multimap}_{\mathtt{ls}}\overset{i}{\multimap}_{\mathtt{dB}} \subseteq \overset{i}{\multimap}_{\mathtt{dB}}\overset{\infty}{\multimap}_{\mathtt{ls}}$.

3) ***Row 2a:*** $\overset{j}{\multimap}_{\mathtt{ls}}\overset{i}{\multimap}_{\mathtt{ls}} \subseteq \overset{i}{\multimap}_{\mathtt{ls}}\overset{j}{\multimap}_{\mathtt{ls}}$. *6)* ***Diagonal 2a:*** $\overset{j}{\multimap}_{\mathtt{dB}}\overset{i}{\multimap}_{\mathtt{ls}} \subseteq \overset{i}{\multimap}_{\mathtt{ls}}\overset{j}{\multimap}_{\mathtt{dB}}$.

7) ***Row 2b:*** $\overset{\infty}{\multimap}_{\mathtt{ls}}\overset{i}{\multimap}_{\mathtt{ls}} \subseteq \overset{i}{\multimap}_{\mathtt{ls}}\overset{\infty}{\multimap}_{\mathtt{ls}} \cup \overset{i}{\multimap}_{\mathtt{ls}}\overset{i+k}{\multimap}_{\mathtt{ls}}\overset{\infty}{\multimap}_{\mathtt{ls}} \cup \overset{i}{\multimap}_{\mathtt{ls}}\overset{\infty}{\multimap}_{\mathtt{ls}}\overset{\infty}{\multimap}_{\mathtt{ls}}$, *with $k \in \mathbb{N}$.*

8) ***Diagonal 2b:*** $\overset{\infty}{\multimap}_{\mathtt{dB}}\overset{i}{\multimap}_{\mathtt{ls}} \subseteq \overset{i}{\multimap}_{\mathtt{ls}}\overset{\infty}{\multimap}_{\mathtt{dB}} \cup \overset{i}{\multimap}_{\mathtt{ls}}\overset{i+k}{\multimap}_{\mathtt{dB}}\overset{\infty}{\multimap}_{\mathtt{dB}} \cup \overset{i}{\multimap}_{\mathtt{ls}}\overset{\infty}{\multimap}_{\mathtt{dB}}\overset{\infty}{\multimap}_{\mathtt{dB}}$, *with $k \in \mathbb{N}$.*

The lemma implies factorization by levels analogous to the one of $\lambda_{\mathtt{sub}}$ (but with respect to different leveled reductions).

▶ **Corollary 7.7** (Factorization by levels)**.** $(\overset{>i}{\multimap}_{\mathtt{dB}}, \overset{i}{\multimap}_{\mathtt{dB}}, \overset{>i}{\multimap}_{\mathtt{ls}}, \overset{i}{\multimap}_{\mathtt{ls}})$ *is a SFS for every $i \in \mathbb{N}$, and so $\overset{\geq i}{\multimap}{}^* \subseteq \overset{i}{\multimap}{}^* \overset{>i}{\multimap}{}^*$.*

Proof. The first axiom (termination) follows from Lemma 7.1. The second from points 1 and 2 of Lemma 7.6. The third form points 3 and 7. The fourth from points 4 and 5. The fifth from points 6 and 8. ◀

We can now easily prove Theorem 7.4, just note that $(\multimap_{\mathtt{dB}\bullet}, \multimap_{\mathtt{dB}\circ}, \multimap_{\mathtt{ls}\bullet}, \multimap_{\mathtt{ls}\circ})$ is nothing but $(\overset{>0}{\multimap}_{\mathtt{dB}}, \overset{0}{\multimap}_{\mathtt{dB}}, \overset{>0}{\multimap}_{\mathtt{ls}}, \overset{0}{\multimap}_{\mathtt{ls}})$. We conclude with a standardization theorem. Define the **linear level** of a term t as the maximum $i \in \mathbb{N}$ s.t. $t = H_i[u]$ for some u.

▶ **Theorem 7.8** (Standardization by levels)**.** *Let $t \to^*_{\lambda_{\mathtt{lsub}}} u$ and k is the linear level of u. Then $t \overset{0}{\multimap}{}^* \overset{1}{\multimap}{}^* \ldots \overset{k-1}{\multimap}{}^* \overset{k}{\multimap}{}^* \overset{\infty}{\multimap}{}^* \to^*_{\mathtt{gc}} u$.*

Proof. Remark that $\overset{\infty}{\multimap}$ and $\to_{\mathtt{gc}}$ cannot change the linear level of a term and that $t \overset{i}{\multimap} v$ for i finite implies that both t and v have linear level at least i. From Lem. 7.2.2 we get that there exists v s.t. $t \to^*_{\neg\mathtt{gc}} v \to^*_{\mathtt{gc}} u$. By the remark v has linear level k, the same of u. By iterating the use of Corollary 7.7 as in the proof of Corollary 3.7 we factor $\to^*_{\neg\mathtt{gc}}$ as $t \overset{0}{\multimap}{}^* \overset{1}{\multimap}{}^* \ldots \overset{k-1}{\multimap}{}^* \overset{k}{\multimap}{}^* \overset{>k}{\multimap}{}^* v$. Now, consider $\overset{>k}{\multimap}{}^*$. By the remark $\overset{>k}{\multimap}{}^*$ has to be a $\overset{\infty}{\multimap}{}^*$ sequence and so we conclude. ◀

Theorem 7.8 provides a sharp decomposition of $\lambda_{\mathtt{lsub}}$-reductions. It is easy to prove that $\overset{j}{\multimap}$ has the diamond property, for every $j \in \mathbb{N}$. However, the theorem can certainly be strengthened, as it says nothing about $\overset{\infty}{\multimap}$ (which does not enjoy the diamond property), whose analysis requires further factorizations, left to future work.

8 Conclusions

We presented an abstract factorization theorem for explicit substitution (ES) calculi, with applications to ES-calculi related to Proof-Nets. The abstract technique is simple. The theorem is applied more than once to every calculus, getting different factorizations, showing the flexibility of the abstract approach. It captures both call-by-name and call-by-value dynamics. We are convinced that the theorem applies to many other calculi, not related to Proof-Nets, despite this verification is left to future work.

We believe that our study of linear head reduction is a contribution: our presentation is simpler than the usual one [14], and the lack of a rewriting characterization for this important notion was a hole in the literature. A more detailed study of the linear substitution calculus, by means of residuals and permutation equivalence is an interesting research direction. It is also interesting to find other rewriting properties which can take advantage of the local termination nature of explicit substitution calculi.

Acknowledgements I want to thank Delia Kesner for many discussions on the subject and for her encouragement to try to publish this work. I also want to thank Flavio C. de Moura, Luca Paolini, Paul-André Melliès, Willem Heijltjes, and the anonymous referees.

References

1 M. Abadi, L. Cardelli, P. L. Curien, and J. J. Levy. Explicit substitutions. *Journal of Functional Programming*, 1:31–46, 1991.

2 B. Accattoli. An abstract factorisation theorem for explicit substitutions (extended version). available at `https://sites.google.com/site/beniaminoaccattoli/tech-report.pdf`.

3 B. Accattoli. *Jumping around the box: graphical and operational studies on Lambda Calculus and Linear Logic.* Ph.D. Thesis, Università di Roma La Sapienza, 2011.

4 B. Accattoli and S. Guerrini. Jumping boxes. representing λ-calculus boxes by jumps. In *CSL*, volume 5771 of *LNCS*, pages 55–70. Springer, 2009.

5 B. Accattoli and D. Kesner. The structural λ-calculus. In *Computer Science Logic (CSL)*, volume 6247 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2010.

6 B. Accattoli and D. Kesner. The permutative -calculus. In *LPAR*, pages 23–36, 2012.

7 B. Accattoli and U. Dal Lago. On the invariance of the unitary cost model for head reduction. Accepted at RTA 2012.

8 B. Accattoli and L. Paolini. Call-by-value solvability, revisited. Accepted to FLOPS 2012, available at `https://sites.google.com/site/beniaminoaccattoli/solvability.pdf`.

9 H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, Amsterdam, revised edition, 1984. (First edition, 1981).

10 R. Di Cosmo and A. Piperno. Expanding extensional polymorphism. In *TLCA*, pages 139–153, 1995.

11 Pierre-Louis Curien, Thérèse Hardin, and Jean-Jacques Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *J. ACM*, 43(2):362–397, 1996.

12 V. Danos, H. Herbelin, and L. Regnier. Game semantics & abstract machines. In *LICS*, pages 394–405, 1996.

13 V. Danos and L. Regnier. Reversible, irreversible and optimal lambda-machines. *Electr. Notes Theor. Comput. Sci.*, 3, 1996.

14 V. Danos and L. Regnier. Head linear reduction. Technical report, 2004.

15 D. de Carvalho, M. Pagani, and L. Tortora de Falco. A semantic measure of the execution time in linear logic. *TCS, Special issue Girard's Festschrift*, 412(20):1884–1902, 2011.

16 T. Ehrhard and L. Regnier. Böhm trees, krivine's machine and the taylor expansion of lambda-terms. In *CiE*, pages 186–197, 2006.

17 A. Geser. *Relative Termination*. Ph.d. thesis, Fakultät fürMathematik und Informatik, Universität Passau, Germany, 1990.

18 Georges Gonthier, Jean-Jacques Lévy, and Paul-André Melliès. An abstract standardisation theorem. In *LICS*, pages 72–81, 1992.

19 H. Herbelin and S. Zimmermann. An operational account of call-by-value minimal and classical lambda-calculus in "natural deduction" form. In *TLCA*, pages 142–156, 2009.

20 R. E. Møgelberg J. Egger and A. Simpson. Enriching an effect calculus with linear types. In *CSL*, pages 240–254, 2009.

21 D. N. Turner J. Maraist, M. Odersky and P. Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Theor. Comput. Sci.*, 228(1-2):175–210, 1999.

22 D. Kesner. The theory of calculi with explicit substitutions revisited. In *CSL*, pages 238–252, 2007.

23 G. Mascari and M. Pedicini. Head linear reduction and pure proof net extraction. *Theor. Comput. Sci.*, 135(1):111–137, 1994.

24 D. Mazza. Pi et lambda. Une étude sur la traduction des lambda-termes dans le pi-calcul. Memoire de DEA (in french), 2003.

25 P.-A. Melliès. Typed lambda-calculi with explicit substitutions may not terminate. In *TLCA*, pages 328–334, 1995.

26 P.-A. Melliès. *Description axiomatique des systèmes de réécriture*. Phd thesis, Université Paris VII, 1996.

27 P.-A. Melliès. A factorisation theorem in rewriting theory. In *Category Theory and Computer Science*, pages 49–68, 1997.

28 P.-A. Melliès. Axiomatic rewriting theory I: A diagrammatic standardization theorem. In *Processes, Terms and Cycles*, pages 554–638, 2005.

29 R. Milner. Local bigraphs and confluence: Two conjectures: (extended abstract). *Electr. Notes Theor. Comput. Sci.*, 175(3):65–73, 2007.

30 M. Takahashi. Parallel reductions in lambda-calculus. *Inf. Comput.*, 118(1):120–127, 1995.

31 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

32 K. Terui. Light affine lambda calculus and polynomial time strong normalization. *Arch. Math. Log.*, 46(3-4):253–280, 2007.

33 P. Tranquilli. *Nets Between Determinism and Nondeterminism*. Ph.d. thesis, Università degli Studi Roma Tre/Université Paris Diderot (Paris 7), 2009.

On the Invariance of the Unitary Cost Model for Head Reduction*

Beniamino Accattoli[1] **and Ugo Dal Lago**[2]

1 **INRIA & LIX (École Polytechnique)**
`beniamino.accattoli@inria.fr`

2 **Università di Bologna & INRIA**
`dallago@cs.unibo.it`

Abstract

The λ-calculus is a widely accepted computational model of higher-order functional programs, yet there is not any direct and universally accepted cost model for it. As a consequence, the computational difficulty of reducing λ-terms to their normal form is typically studied by reasoning on concrete implementation algorithms. In this paper, we show that when head reduction is the underlying dynamics, the unitary cost model is indeed invariant. This improves on known results, which only deal with weak (call-by-value or call-by-name) reduction. Invariance is proved by way of a *linear* calculus of explicit substitutions, which allows to nicely decompose any head reduction step in the λ-calculus into more elementary substitution steps, thus making the combinatorics of head-reduction easier to reason about. The technique is also a promising tool to attack what we see as the main open problem, namely understanding for which *normalizing* strategies the unitary cost model is invariant, if any.

1998 ACM Subject Classification F.4.1 Mathematical Logic – Lambda calculus and related systems, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases lambda calculus, cost models, explicit substitutions, implicit computational complexity

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.22

Category Regular Research Paper

1 Introduction

Giving an estimate of the amount of time T needed to execute a program is a natural refinement of the termination problem, which only requires to decide whether T is either finite or infinite. The shift from termination to complexity analysis brings more informative outcomes at the price of an increased difficulty. In particular, complexity analysis depends much on the chosen computational model. Is it possible to express such estimates in a way which is independent from the specific machine the program is run on? An answer to this question can be given following computational complexity, which classifies algorithms (and functions) based on the amount of time (or space) they consume when executed by *any* abstract device endowed with a *reasonable* cost model, depending on the size of input. When can a cost model be considered reasonable? The answer lies in the so-called invariance thesis [23]: any time cost model is reasonable if it is polynomially related to the (standard) one of Turing machines.

* This work was partially supported by the ARC INRIA project "ETERNAL".

If programs are expressed as rewrite systems (e.g. as first-order TRSs), an abstract but effective way to execute programs, rewriting itself, is always available. As a consequence, a natural time cost model turns out to be *derivational complexity*, namely the (maximum) number of rewrite steps which can possibly be performed from the given term. A rewriting step, however, may not be an atomic operation, so derivational complexity is not by definition invariant. For first-order TRSs, however, derivational complexity has been recently shown to be an invariant cost model, by way of term graph rewriting [12, 8, 9].

The case of λ-calculus is definitely more delicate: if β-reduction is weak, i.e., if it cannot take place in the scope of λ-abstractions, one can see λ-calculus as a TRS and get invariance by way of the already cited results [11], or by other means [21]. But if one needs to reduce "under lambdas" because the final term needs to be in normal form (e.g., when performing type checking in dependent type theories), no invariance results are known at the time of writing.

In this paper we give a partial solution to this problem, by showing that the unitary cost model is indeed invariant for the λ-calculus endowed with *head reduction*, in which reduction *can* take place in the scope of λ-abstractions, but *can only* be performed in head position. Our proof technique consists in implementing head reduction in a calculus of explicit substitutions.

Explicit substitutions were introduced to close the gap between the theory of λ-calculus and implementations [1]. Their rewriting theory has also been studied in depth, after Melliès showed the possibility of pathological behaviors [15]. Starting from graphical syntaxes, a new *at a distance* approach to explicit substitutions has recently been proposed [6]. The new formalisms are simpler than those of the earlier generation, and another thread of applications — to which this paper belongs — also started: new results on λ-calculus have been proved by means of explicit substitutions [6, 7].

In this paper we use the *linear-substitution calculus* $\Lambda_{[\cdot]}$, a slight variation over a calculus of explicit substitutions introduced by Robin Milner [17]. The variation is inspired by the structural λ-calculus [6]. We study in detail the relation between λ-calculus head reduction and *linear head reduction* [14], the head reduction of $\Lambda_{[\cdot]}$, and prove that the latter is at most quadratically longer than the former. This is proved without any termination assumption, by a detailed rewriting analysis.

To get the Invariance Theorem, however, other ingredients are required:

1. *The Subterm Property.* Linear head reduction has a property not enjoyed by head β-reduction: linear substitutions along a reduction $t \multimap^* u$ duplicate subterms of t only. It easily follows that $\multimap$-steps can be simulated by Turing machines in time polynomial in the size of t and the length of $\multimap^*$. This is explained in Section 3.
2. *Compact representations.* Explicit substitutions, decomposing β-reduction into more atomic steps, allow to take advantage of sharing and thus provide compact representations of terms, avoiding the exponential blowups of term size happening in plain λ-calculus. Is it reasonable to use these compact representations of λ-terms? We answer affirmatively, by exhibiting a dynamic programming algorithm for checking equality of terms with explicit substitutions modulo unfolding, and proving it to work in polynomial time in the size of the involved compact representations. This is the topic of Section 5.
3. *Head simulation of Turing machines.* We also provide the simulation of Turing machines by λ-terms. We give a new encoding of Turing machines, since the known ones do not work with *head* β-reduction, and prove it induces a polynomial overhead. Some details of the encoding are given in Section 6.

We emphasize the result for head β-reduction, but our technical detour also proves invariance for linear head reduction. To our knowledge, we are the first ones to use the fine granularity of explicit substitutions for complexity analysis. Many calculi with bounded complexity (e.g. [22]) use `let`-constructs, an avatar of explicit substitutions, but they do not take advantage of the refined dynamics, as they always use big-steps substitution rules.

To conclude, we strongly believe that the main contribution of this paper lies in the technique rather than in the invariance result. Indeed, the main open problem in this area, namely the invariance of the unitary cost model for any *normalizing* strategy remains open but, as we argue in Section 4, seems now within reach.

An extended version of this paper with more detailed proofs is available [4].

2 λ-Calculus and Cost Models: an Informal Account

Consider the pure, untyped, λ-calculus. Terms can be variables, abstractions or applications and computation is modeled by β-reduction. Once a reduction strategy is fixed, one could be tempted to make *time* and *reduction steps* to correspond: firing a β-redex requires one time instant (or, equivalently, a finite number of time instants) and thus the number of reduction steps to normal form could be seen as a measure of its time complexity. This would be very convenient, since reasoning on the complexity of normalization could be done this way directly on λ-terms. However, doing so one could in principle risk to be too optimistic about the complexity of obtaining the normal form of a term t, given t as an input. This section will articulate on this issue by giving some examples and pointers to the relevant literature.

Consider the sequence of λ-terms defined as follows, by induction on a natural number n (where u is the lambda term yxx): $t_0 = u$ and for every $n \in \mathbb{N}$, $t_{n+1} = (\lambda x.t_n)u$. t_n has size linear in n, and t_n rewrites to its normal form r_n in exactly n steps, following a leftmost-outermost strategy:

$$
\begin{aligned}
&t_0 \equiv u \equiv r_0;\\
&t_1 \to yuu \equiv yr_0r_0 \equiv r_1;\\
&t_2 \to (\lambda x.t_0)(yuu) \equiv (\lambda x.u)r_1 \to yr_1r_1 \equiv r_2;\\
&\quad\vdots
\end{aligned}
$$

For every n, however, r_{n+1} contains two copies of r_n, hence the size of r_n is *exponential* in n. As a consequence, if we stick to the leftmost-outermost strategy and if we insist on normal forms to be represented explicitly, without taking advantage of sharing, the unitary cost model *is not* invariant: in a linear number of β-steps we reach an object which cannot even be written down in polynomial time.

One may wonder whether this problem is due to the specific, inefficient, adopted strategy. However, it is quite easy to rebuild a sequence of terms exhibiting the same behavior along an innermost strategy: if $s = \lambda y.yxx$, define v_0 to be just $\lambda x.s$, for every $n \in \mathbb{N}$, v_{n+1} to be $\lambda x.v_n s$, and consider $v_n(\lambda x.x)$. Actually, there *are* invariant cost-models for the λ-calculus even if one wants to obtain the normal form in an explicit, linear format, like the difference cost model [10]. But they pay a price for that: they do not attribute a constant weight to each reduction step. Then, another natural question arises: is it that the gap between the unitary cost model and the real complexity of reducing terms is only due to a *representation* problem? In other words, could we take advantage of a shared representation of terms, even if only to encode λ-terms (and normal forms in particular) in a compact way?

The literature offers some positive answers to the question above. In particular, the unitary cost model can be proved to be invariant for both call-by-name and call-by-value

λ-calculi, as defined by Plotkin [19]. In one way or another, the mentioned results are based on sharing subterms, either by translating the λ-calculus to a TRS [11] or by going through abstract machines [21]. Plotkin's calculi, however, are endowed with *weak* notions of reduction, which prevent computation to happen in the scope of a λ-abstraction. And the proposed approaches crucially rely on that.

The question now becomes the following: is it possible to prove the invariance of the unitary cost model for some *strong* notion of reduction? This paper gives a first, positive answer to this question by proving the number of β-reduction steps to be an invariant cost model for *head* reduction, in which one *is* allowed to reduce in the scope of λ-abstractions, but evaluation stops on *head* normal forms.

We are convinced that the exponential blowup in the examples above is, indeed, only due to the λ-calculus being a very inefficient *representation* formalism. Following this thesis we use terms with explicit substitutions as compact representations: our approach, in contrast to other ones, consists in using sharing (here under the form of explicit substitutions) only to obtain compactness, and not to design some sort of optimal strategy reducing shared redexes. Actually, we follow the opposite direction: the leftmost-outermost strategy — being standard — can be considered as the maximally *non-sharing* strategy. How much are we losing limiting ourselves to head reduction? Not so much: in Section 6 we show an encoding of Turing machines for which the normal form is reached by head-reduction only. Moreover, from a denotational semantics point of view head-normal forms — and not full normal forms — are the right notion of result for β-reduction.

The next two sections introduce explicit substitutions and prove that the length of their head strategy is polynomially related to the length of head β-reduction. In other words, the switch to compact representations does not affect the cost model in any essential way.

3 Linear Explicit Substitutions

First of all, we introduce the λ-calculus. Its terms are given by the grammar:

$$t, u, r ::= x \mid t\, t \mid \lambda x.t$$

and its reduction rule $\rightarrow_\beta$ is defined as the context closure of $(\lambda x.t)\ u \mapsto_\beta t\{x/u\}$. $\mathcal{T}_\lambda$ is the set of all terms of the λ-calculus. We will mainly work with head reduction, instead of full β-reduction. We define head reduction as follows. Let an *head context* $\hat{H}$ be defined by:

$$\hat{H} ::= [\cdot] \mid \hat{H}\ t \mid \lambda x.\hat{H}.$$

Then define *head reduction* $\rightarrow_{\mathtt{h}}$ as the closure by head contexts of $\mapsto_\beta$. Our definition of head reduction is slightly more liberal than the usual one. Indeed, it is non-deterministic, for instance:

$$(\lambda x.I)\ t\ {}_{\mathtt{h}}\!\leftarrow (\lambda x.(I\ I))\ t \rightarrow_{\mathtt{h}} I\ I.$$

Usually only one of the two redexes would be considered an head redex. However, this non-determinism is harmless, since one easily proves that $\rightarrow_{\mathtt{h}}$ has the diamond property. Reducing $\rightarrow_{\mathtt{h}}$ in an outermost way we recover the usual notion of head reduction, so our approach gains in generality without loosing any property of head reduction. Our notion is motivated by the corresponding notion of head reduction for explicit substitutions, which is easier to manage in this more general approach.

The calculus of explicit substitutions we are going to use is a minor variation over a simple calculus introduced by Milner [17]. The grammar is standard:

$$t, u, r ::= x \mid t\, t \mid \lambda x.t \mid t[x/t].$$

$$\begin{array}{lll} (\lambda x.t)\mathtt{L}\; u & \mapsto_{\mathtt{dB}} & t[x/u]\mathtt{L} \\ C[x][x/u] & \mapsto_{\mathtt{ls}} & C[u][x/u] \\ t[x/u] & \mapsto_{\mathtt{gc}} & t \qquad\qquad \text{if } x \notin \mathtt{fv}(t) \end{array}$$

Figure 1 $\Lambda_{[\cdot]}$ Rewriting Rules.

The term $t[x/u]$ is an *explicit substitution.* $\mathcal{T}$ is the set of terms with explicit substitutions. Observe that $\mathcal{T}_\lambda \subset \mathcal{T}$. Both constructors $\lambda x.t$ and $t[x/u]$ bind x in t. We note $\mathtt{L}$ a possibly empty list of explicit substitutions $[x_1/u_1]\dots[x_k/u_k]$. Contexts are defined by:

$$C, D, E, F ::= [\cdot] \mid C\; t \mid t\; C \mid \lambda x.C \mid C[x/t] \mid t[x/C].$$

We note $C[t]$ the usual operation of substituting $[\cdot]$ in C possibly capturing free variables of t. We will often use expressions like $C[x][x/u]$ where it is implicitly assumed that C does not capture x. The *linear-substitution calculus* $\Lambda_{[\cdot]}$ is given by the rewriting rules $\to_{\mathtt{dB}}$, $\to_{\mathtt{ls}}$ and $\to_{\mathtt{gc}}$, defined as the context closures of the rules $\mapsto_{\mathtt{dB}}$, $\mapsto_{\mathtt{ls}}$ and $\mapsto_{\mathtt{gc}}$ in Figure 1. We also use the notation $\to_{\Lambda_{[\cdot]}} = \to_{\mathtt{dB}} \cup \to_{\mathtt{ls}} \cup \to_{\mathtt{gc}}$ and $\to_{\mathtt{s}} = \to_{\mathtt{ls}} \cup \to_{\mathtt{gc}}$. Rule $\to_{\mathtt{dB}}$ acts at a *distance*: the function $\lambda x.t$ and the argument u can interact even if there is $\mathtt{L}$ between them. This is motivated by the close relation between $\Lambda_{[\cdot]}$ and graphical formalisms as proof-nets and $\lambda\mathtt{j}$-dags [3, 5], and is also the difference with Milner's presentation of $\Lambda_{[\cdot]}$ [17].

The linear-substitution calculus enjoys all properties required to explicit substitutions calculi, obtained by easy adaptations of the proofs for $\lambda\mathtt{j}$ in [6]. Moreover, it is confluent and preserves β-strong normalization. In particular, $\to_{\mathtt{s}}$ is a strongly normalizing and confluent relation.

Given a term t with explicit substitutions, its normal form with respect to $\to_{\mathtt{s}}$ is a λ-term, noted $t{\downarrow}$, called the *unfolding* of t and verifying the following equalities:

$$(t\; u){\downarrow} = t{\downarrow}\, u{\downarrow}; \qquad (\lambda x.t){\downarrow} = \lambda x.t{\downarrow}; \qquad (t[x/u]){\downarrow} = t{\downarrow}\{x/u{\downarrow}\}.$$

Another useful property is the so-called *full-composition*, which states that any explicit substitution can be reduced to its implicit form independently from the other substitutions in the term, formally $t[x/u] \to_{\mathtt{s}}^{*} t\{x/u\}$. Last, $\Lambda_{[\cdot]}$ simulates λ-calculus ($t \to_\beta u$ implies $t \to_{\Lambda_{[\cdot]}}^{*} u$) and reductions in $\Lambda_{[\cdot]}$ can be projected on λ-calculus via unfolding ($t \to_{\Lambda_{[\cdot]}} u$ implies $t{\downarrow} \to_\beta^{*} u{\downarrow}$).

The calculus $\Lambda_{[\cdot]}$ has a strong relation with proof-nets and linear logic: it can be mapped to Danos' and Regnier's pure proof-nets [20] or to $\lambda\mathtt{j}$-dags [5]. The rule $\to_{\mathtt{dB}}$ corresponds to proof-nets multiplicative cut-elimination, $\to_{\mathtt{ls}}$ to the cut-elimination rule between ! (every substitution is in a !-box) and contraction, $\to_{\mathtt{gc}}$ to the cut-elimination rule between ! and weakening. The case of a cut between ! and dereliction is handled by $\to_{\mathtt{ls}}$, as if cut derelictions were always contracted with a weakening.

3.1 Linear Head Reduction, the Subterm Property and Shallow Terms

In this paper, we mainly deal with a specific notion of reduction for $\Lambda_{[\cdot]}$, called *linear head reduction* [14, 13], and related to the representation of λ-calculus into linear logic proof-nets. In order to define it we need the notion of head context for explicit substitutions.

▶ **Definition 3.1** (head context). *Head contexts* are defined by the following grammar:

$$H ::= [\cdot] \mid H\; t \mid \lambda x.H \mid H[x/t].$$

The fundamental property of an head context H is that the hole cannot be duplicated nor erased. In terms of linear logic proof-nets, the head hole is not contained in any box (since boxes are associated with the arguments of applications and with explicit substitutions). We now need to consider a modified version of $\mapsto_{\mathtt{ls}}$:

$$H[x][x/u] \hat{\multimap}_{\mathtt{ls}} H[u][x/u].$$

Now, let $\multimap_{\mathtt{dB}}$ (resp. $\multimap_{\mathtt{ls}}$) be the closure by head contexts of $\mapsto_{\mathtt{dB}}$ (resp. $\hat{\multimap}_{\mathtt{ls}}$). Last, define *head linear reduction* $\multimap$ as $\multimap_{\mathtt{dB}} \cup \multimap_{\mathtt{ls}}$. Please notice that $\multimap$ can reduce under λ, for instance $\lambda y.(H[x][x/u]) \multimap_{\mathtt{ls}} \lambda y.(H[u][x/u])$. Our definition of $\multimap$ gives a non-deterministic strategy, but its non-determinism is again harmless: a simple case analysis shows that $\multimap$ has the diamond property.

In [2] it is proved that linear head reduction has the same factorization property enjoyed by head reduction in λ-calculus: if $t \to^*_{\Lambda_{[\cdot]}} u$ then $t \multimap^* \Rightarrow^*$, where $\Rightarrow$ is the complement of $\to_{\Lambda_{[\cdot]}}$ with respect to $\multimap$ (*i.e.* $\Rightarrow := \to_{\Lambda_{[\cdot]}} \setminus \multimap$).

A term u is a *box-subterm* of a term t (resp. of a context C) if t (resp. C) has a subterm of the form $r\ u$ or of the form $r[x/u]$ for some r.

▶ Remark. By definition of head-contexts, $[\cdot]$ is not a box-subterm of $H[\cdot]$, and there is no box-subterm of $H[\cdot]$ which has $[\cdot]$ as a subterm.

The following fundamental property of head linear reduction is folklore among specialists, but to our knowledge it has never appeared in print before.

▶ **Proposition 3.2** (Subterm Property). *If $t \multimap^* u$ and r is a box-subterm of u, then r is a box-subterm of t.*

The aforementioned proposition is a key point in our study of cost models. Linear head substitution steps duplicate sub-terms, but the Subterm Property guarantees that only subterms of the initial term t are duplicated, and thus each step can be implemented in time polynomial in the size of t, which is the size of the input, the fundamental parameter for complexity analysis. This is in sharp contrast with what happens in the λ-calculus, where the cost of a β-reduction step is not even polynomially related to the size of the initial term.

Proof. Let $t \multimap^k u$. We proceed by induction on k. Suppose that $k > 0$. Then $t \multimap^* v \multimap u$ and by *i.h.* any box-subterm of v is a box subterm of t, so it is enough to show that any box-subterm of u is a box-subterm of v. By induction on $v \multimap u$. The case $v = (\lambda x.r)\mathtt{L}\ s \multimap r[x/s]\mathtt{L}$ is trivial. If $v = H[x][x/s] \multimap H[s][x/s]$ by the previous remark the plug of s in $H[\cdot]$ does not create any new box-subterm, nor modify a box-subterm of $H[\cdot]$. And obviously any box-subterm of s is a box-subterm of v. The inductive cases follow from the *i.h.* and the previous remark. ◀

The subterm property does not only allow to bound the cost of implementing any reduction step, but also to bound the size of intermediate terms:

▶ **Corollary 3.3.** *There is a polynomial $p : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ such that if $t \multimap^k u$ then $|u| \leq p(k, |t|)$.*

Consider a reduction $t \multimap^* u$ where t is a λ-term. Another consequence of the Subterm Property is that for every explicit substitution occurring in u, the substituted term is a λ-term. This is another strong property to be used in the analysis of the next section.

▶ **Definition 3.4** (Shallow Terms). A $\Lambda_{[\cdot]}$-term t is *shallow* if whenever $t = C[u[x/r]]$ then r is a λ-term.

▶ **Corollary 3.5.** *Let t be a λ-term and $t \multimap^* u$. Then u is shallow.*

4 On the Relation Between Λ and $\Lambda_{[\cdot]}$

In this section, linear explicit substitutions will be shown to be an efficient way to implement head reduction. We will proceed by proving three auxiliary results separately:

1. We show that any $\multimap$-reduction ρ projects via unfolding to a $\rightarrow_{\mathtt{h}}$-reduction $\rho\!\downarrow$ having as length exactly the number of $\multimap_{\mathtt{dB}}$ steps in ρ; this is the topic of Section 4.1;
2. We show the converse relation, *i.e.* that any $\rightarrow_{\mathtt{h}}$-reduction ρ can be simulated by a $\multimap$-reduction having as many $\multimap_{\mathtt{dB}}$-steps as the the steps in ρ, followed by unfolding; this is in Section 4.2;
3. We show that in any $\multimap$-reduction ρ the number of $\multimap_{\mathtt{ls}}$-steps is $\mathcal{O}(|\rho|_{\mathtt{dB}}^2)$ where $|\rho|_{\mathtt{dB}}$ is the number of $\multimap_{\mathtt{dB}}$ steps in ρ. By the previous two points, there is a quadratic — and thus polynomial — relation between $\rightarrow_{\mathtt{h}}$-reductions and $\multimap$-reduction from a given term; all this is explained in Section 4.3.

4.1 Projection of $\multimap$ on $\rightarrow_{\mathtt{h}}$

The first thing one needs to prove about head-linear reduction is whether it is a *sound* way to implement head reduction. This is proved by relatively standard techniques, and requires the following auxiliary lemma, whose proof is by induction on $t \rightarrow_{\mathtt{h}} u$:

▶ **Lemma 4.1.** *Let $t \in \mathcal{T}_\lambda$. If $t \rightarrow_{\mathtt{h}} u$ then $t\{x/r\} \rightarrow_{\mathtt{h}} u\{x/r\}$.*

▶ **Lemma 4.2** (Projection of $\multimap$ on $\rightarrow_{\mathtt{h}}$). *Let $t \in \mathcal{T}$. If $\rho : t \multimap^k u$ then $t\!\downarrow \rightarrow_{\mathtt{h}}^n u\!\downarrow$ and $n = |\rho|_{\mathtt{dB}} \leq k$.*

Proof. By induction on k. If $k = 0$ it is trivial, so let $k > 0$ and $t \multimap^{n-1} r \multimap u$. Let τ be the reduction $t \multimap^{n-1} r$. By *i.h.* we get $t\!\downarrow \rightarrow_{\mathtt{h}}^m r\!\downarrow$ and $m = |\tau|_{\mathtt{dB}} \leq k - 1$. Now consider $r \multimap u$. There are two cases. If $r \multimap_{\mathtt{ls}} u$ then $r\!\downarrow = u\!\downarrow$, by definition of $(\cdot)\!\downarrow$, as the normal form of $\rightarrow_{\mathtt{s}}$ (which contains $\multimap_{\mathtt{ls}}$). If $r \multimap_{\mathtt{dB}} u$ then $r = H[(\lambda x.v)\mathtt{L}\ s] \multimap H[v[x/s]\mathtt{L}] = u$. We prove that $r\!\downarrow \rightarrow_{\mathtt{h}} u\!\downarrow$, from which it follows that $t\!\downarrow \rightarrow_{\mathtt{h}}^{m+1} u\!\downarrow$, where $m + 1 = |\tau|_{\mathtt{dB}} + 1 = |\rho|_{\mathtt{dB}} \leq k$. We go by induction on H. We use the following notation: if $t = u\mathtt{L}$ and $\mathtt{L} = [y_1/w_1] \dots [y_m/w_m]$ then we write $\sigma_{\mathtt{L}}$ for $\{y_1/w_1\!\downarrow\} \dots \{y_m/w_m\!\downarrow\}$, thus we can write $t\!\downarrow = u\!\downarrow\!\sigma_{\mathtt{L}}$. Cases:

- $H = [\cdot]$. Then:

$$\begin{aligned} r\!\downarrow =& ((\lambda x.v)\mathtt{L})\!\downarrow s\!\downarrow = (\lambda x.v\!\downarrow)\sigma_{\mathtt{L}}\ s\!\downarrow = (\lambda x.v\!\downarrow\!\sigma_{\mathtt{L}})\ s\!\downarrow \\ \rightarrow_{\mathtt{h}}& v\!\downarrow\!\sigma_{\mathtt{L}}\{x/s\!\downarrow\} \overset{*}{=} v\!\downarrow\!\{x/s\!\downarrow\}\sigma_{\mathtt{L}} = (v[x/s])\!\downarrow\!\sigma_{\mathtt{L}} = (v[x/s]\mathtt{L})\!\downarrow = u\!\downarrow. \end{aligned}$$

  Step (*) holds because $x \notin \mathtt{fv}(w_i)$ for $i \in \{1, \dots, m\}$.
- $H = J[y/w]$. Then by *i.h.* and Lemma 4.1 we get $r\!\downarrow = (J[(\lambda x.v)\mathtt{L}\ s])\!\downarrow\!\{y/w\!\downarrow\} \rightarrow_{\mathtt{h}} (J[v[x/s]\mathtt{L}])\!\downarrow\!\{y/w\!\downarrow\} = u\!\downarrow$
- The cases $H = J\ w$ and $H = \lambda y.J$ follow from the *i.h.*

◀

4.2 Projection of $\rightarrow_{\mathtt{h}}$ on $\multimap$

Here we map head β-steps to head linear steps followed by unfolding. In other words, we prove that head-linear reduction is not only a sound, but also a *complete* way to implement head reduction. This section is going to be technically more involved than the previous one. First of all, we show that a single head step can be simulated by a step of $\multimap_{\mathtt{dB}}$ followed by unfolding, which is straightforward:

▶ **Lemma 4.3** (Head Simulation). *Let t be a λ-term. If $t \rightarrow_{\mathtt{h}} u$, then $t \multimap_{\mathtt{dB}} \rightarrow_{\mathtt{s}}^* u$.*

Proof. By induction on $t \to_{\mathtt{h}} u$. The base case: if $(\lambda x.u)\ r \to_{\mathtt{h}} u\{x/r\}$ then $(\lambda x.u)\ r \multimap_{\mathtt{dB}} u[x/r]$ and $u[x/r] \to_{\mathtt{s}}^{*} u\{x/r\}$ by full composition. The inductive cases follows by the *i.h.*. ◀

We are now going to show that a sequence of $\to_{\mathtt{ls}}$ reduction steps can be factored into some head-linear substitutions, followed by some linear substitutions under non-head contexts. This allows to refine Lemma 4.3. Define $\Rightarrow_{\mathtt{s}}$ as the relation $\to_{\mathtt{s}} \setminus \multimap_{\mathtt{ls}}$, *i.e.* $\Rightarrow_{\mathtt{s}}$ reduces non-head-linear substitution redexes. Moreover, define the *linear unfolding* $t\downharpoonleft$ of t as the normal form of t with respect to $\multimap_{\mathtt{ls}}$ (which exists since $\multimap_{\mathtt{ls}} \subseteq \to_{\mathtt{ls}}$ and $\to_{\mathtt{ls}}$ terminates, and it is unique because $\multimap_{\mathtt{ls}}$ is deterministic).

Now, we can prove that any $\to_{\mathtt{h}}$ step is simulated in $\multimap_{\mathtt{dB}}\multimap_{\mathtt{ls}}^{*}\Rightarrow_{\mathtt{s}}^{*}$ (actually, in such a sequence there can be at most one $\multimap_{\mathtt{ls}}$ step):

▶ **Lemma 4.4** (Unfolding Factorization). *The following swaps hold:*

1. $\Rightarrow_{\mathtt{s}}\multimap_{\mathtt{ls}} \subseteq \multimap_{\mathtt{ls}}^{+}\Rightarrow_{\mathtt{s}}^{+}$, *precisely:* $\Rightarrow_{\mathtt{s}}\multimap_{\mathtt{ls}} \subseteq \multimap_{\mathtt{ls}}\Rightarrow_{\mathtt{s}} \cup \multimap_{\mathtt{ls}}\multimap_{\mathtt{ls}}\Rightarrow_{\mathtt{s}} \cup \multimap_{\mathtt{ls}}\Rightarrow_{\mathtt{s}}\Rightarrow_{\mathtt{s}}$.
2. $\to_{\mathtt{ls}}^{*} \subseteq \multimap_{\mathtt{ls}}^{*}\Rightarrow_{\mathtt{s}}^{*}$, *and in particular* $t \multimap_{\mathtt{ls}}^{*} t\downharpoonleft \Rightarrow_{\mathtt{s}}^{*} t\downarrow$.

Proof. 1) Formally, the proof is by induction on $t \Rightarrow_{\mathtt{s}} u$. Informally, $\Rightarrow_{\mathtt{s}}$ cannot create, duplicate or alter the head nature of an $\multimap_{\mathtt{ls}}$-step, so that the second step in $t \Rightarrow_{\mathtt{s}}\multimap_{\mathtt{ls}} r$ can be traced back to a unique $\multimap_{\mathtt{ls}}$ redex. Now, there are two cases: either the two redexes simply permute or the preponement of $\multimap_{\mathtt{ls}}$ duplicate the redex reduced by $\Rightarrow_{\mathtt{s}}$. The second case splits in two subcases, of which we just give two examples:

$$\begin{tikzcd}
x[x/y][y/z] \arrow[d, "\multimap_{\mathtt{ls}}"] & \Rightarrow_{\mathtt{s}} \arrow[dl, rightsquigarrow] & x[x/z][y/z] \arrow[d, "\multimap_{\mathtt{ls}}"] \\
y[x/y][y/z] \arrow[r, "\multimap_{\mathtt{ls}}"] & z[x/y][y/z] \arrow[r, "\Rightarrow_{\mathtt{s}}"] & z[x/z][y/z]
\end{tikzcd}$$

$$\begin{tikzcd}
x[x/y\ y][y/z] \arrow[d, "\multimap_{\mathtt{ls}}"] & \Rightarrow_{\mathtt{s}} \arrow[dl, rightsquigarrow] & x[x/y\ z][y/z] \arrow[d, "\multimap_{\mathtt{ls}}"] \\
(y\ y)[x/y\ y][y/z] \arrow[r, "\Rightarrow_{\mathtt{s}}"] & (y\ z)[x/y\ y][y/z] \arrow[r, "\Rightarrow_{\mathtt{s}}"] & (y\ z)[x/y\ z][y/z]
\end{tikzcd}$$

2) There is an abstract lemma (see [4]) which says that if $\to_2\to_1 \subseteq \to_1^{+}\to_2^{*}$ and $\to_1$ is strongly normalizing then $(\to_1 \cup \to_2)^{*} \subseteq \to_1^{*}\to_2^{*}$. Now, taking $\to_1$ to be $\multimap_{\mathtt{ls}}$, $\to_2$ to be $\Rightarrow_{\mathtt{s}}$ and since $\multimap_{\mathtt{ls}}$ is strongly normalising we get $\to_{\mathtt{ls}}^{*} = (\multimap_{\mathtt{ls}} \cup \Rightarrow_{\mathtt{s}})^{*} \subseteq \multimap_{\mathtt{ls}}^{*}\Rightarrow_{\mathtt{s}}^{*}$. ◀

We know that a $\to_{\mathtt{h}}$ step is simulated by a sequence of the form $\multimap_{\mathtt{dB}}\multimap_{\mathtt{ls}}^{*}\Rightarrow_{\mathtt{s}}^{*} \subseteq \multimap^{*}\Rightarrow_{\mathtt{s}}^{*}$. Consider two (or more) $\to_{\mathtt{h}}$-steps. They are simulated by a sequence of the form $\multimap^{*}\Rightarrow_{\mathtt{s}}^{*}\multimap^{*}\Rightarrow_{\mathtt{s}}^{*}$, while we would like to obtain $\multimap^{*}\Rightarrow_{\mathtt{s}}^{*}$. What we need to do is to prove that a sequence of the form $\Rightarrow_{\mathtt{s}}^{*}\multimap_{\mathtt{dB}}$ can always be reorganized as a sequence $\multimap_{\mathtt{dB}}\Rightarrow_{\mathtt{s}}^{*}$:

▶ **Lemma 4.5.** *The following inclusions hold:*

1. $\Rightarrow_{\mathtt{s}}\multimap_{\mathtt{dB}} \subseteq \multimap_{\mathtt{dB}}\Rightarrow_{\mathtt{s}}$.
2. $\Rightarrow_{\mathtt{s}}^{*}\multimap_{\mathtt{dB}} \subseteq \multimap_{\mathtt{dB}}\Rightarrow_{\mathtt{s}}^{*}$.

Proof. 1) By induction on $t \Rightarrow_{\mathtt{s}} u$. The idea is that $\Rightarrow_{\mathtt{s}}$ cannot create, duplicate nor alter the head nature of $\multimap_{\mathtt{dB}}$ redexes, therefore $\multimap_{\mathtt{dB}}$-step can be preponed. Conversely, $\multimap_{\mathtt{dB}}$-steps cannot erase, duplicate nor alter the non-head nature of $\Rightarrow_{\mathtt{s}}$ redexes, so the two steps commute. 2) Let $t \Rightarrow_{\mathtt{s}}^{k}\multimap_{\mathtt{dB}} u$. By induction on k using point 1 and a standard diagram chasing. ◀

The next lemma is the last brick for the projection.

▶ **Lemma 4.6.** *If $t\downarrow \to_{\mathtt{h}} u$ then there exists r s.t. $t\downharpoonleft \multimap_{\mathtt{dB}} r \to_{\mathtt{ls}}^{*} u$.*

Proof. By Lemma 4.3, we get $t\downarrow \multimap\to_{\mathtt{ls}}^{*} u$. By Lemma 4.4, $t \to_{\mathtt{ls}}^{*} t\downarrow$ factors as $t \multimap_{\mathtt{ls}}^{*} t\downharpoonleft \Rightarrow_{\mathtt{s}}^{*} t\downarrow$, and so we get $t\downharpoonleft \Rightarrow_{\mathtt{s}}^{*}\multimap\to_{\mathtt{ls}}^{*} u$. By Lemma 4.5.2, we get $t\downharpoonleft \multimap\Rightarrow_{\mathtt{s}}^{*}\to_{\mathtt{ls}}^{*} u$, *i.e.* $t\downharpoonleft \multimap\to_{\mathtt{ls}}^{*} u$. ◀

We can then conclude with the result which gives the name to this section:

▶ **Lemma 4.7** (Projection of $\to_{\mathtt{h}}$ on $\multimap$). *Let t be a λ-term. If $t \to_{\mathtt{h}}^{k} u$ then there exists a reduction ρ s.t. $\rho : t \multimap^* r$, with $r \to_{\mathtt{ls}}^* u$ and $|\rho|_{\mathtt{dB}} = k$.*

Proof. By induction on k. The case $k = 0$ is trivial, so let $k > 0$ and $t \to_{\mathtt{h}}^{k-1} s \to_{\mathtt{h}} u$. By *i.h.* there exists a reduction τ s.t. $\tau : t \multimap^* v$, $v \to_{\mathtt{ls}}^* s$ and $|\tau|_{\mathtt{dB}} = k - 1$. Since s is a λ-term we have that $v{\downarrow} = s$ and $v{\downarrow} \to_{\mathtt{h}} u$. By lemma 4.6 there exists r s.t. $v{\downarrow_{\multimap}} \multimap_{\mathtt{dB}} r \to_{\mathtt{ls}}^* u$. Moreover, $v \multimap_{\mathtt{ls}}^* v{\downarrow_{\multimap}}$; call γ this reduction. Let ρ be the reduction obtained by concatenating $\tau : t \multimap^* v$, $\gamma : v \multimap_{\mathtt{ls}}^* v{\downarrow_{\multimap}}$ and $v{\downarrow_{\multimap}} \multimap_{\mathtt{dB}} r$. We have $|\rho|_{\mathtt{dB}} = |\tau|_{\mathtt{dB}} + 1 = k$ and $r \to_{\mathtt{ls}}^* u$, and so we conclude. ◀

This section and the previous one proved for $\Lambda_{[\cdot]}$ results proved by more abstract means by Melliès in the context of the $\lambda\sigma$-calculus in [16][1]. The linear substitution calculus is simpler than the $\lambda\sigma$-calculus, which is why our analysis — developed independently — is simpler (but also less sophisticated and general) than Melliès'[2].

4.3 Quadratic Relation

The last two sections proved that head reduction can be seen as head linear reduction followed by unfolding, and that the number of head steps is exactly the number of $\multimap_{\mathtt{dB}}$-steps in the corresponding head linear reduction. To conclude that head and head linear reduction are polynomially related, we need to show that the number of $\multimap_{\mathtt{ls}}$-steps in a linear head reduction ρ is polynomially related to the number of $\multimap_{\mathtt{dB}}$-steps in ρ.

We do it by giving a precise estimate of the maximal length of a $\multimap_{\mathtt{ls}}$ reduction from a given term. Intuition tells us that any reduction $t \multimap_{\mathtt{ls}}^* u$ cannot be longer than the number of explicit substitutions in t (number noted $\mathtt{es}(t)$), since any substitution in t can act at most once on the head variable. However, a formal proof of this fact is not completely immediate, and requires to introduce a measure and prove some (easy) lemmas.

The idea is to statically count the length of the maximum chain of substitutions on the head, and to show that this number decreases at each head linear substitution step. Let us give an example. Consider the reduction:

$$t = (x\ y)[x/y\ r][y/u] \multimap_{\mathtt{ls}} ((y\ r)\ y)[x/y\ r][y/u] \multimap_{\mathtt{ls}} ((u\ r)\ y)[x/y\ r][y/u].$$

It is easy to establish statically on t that $[y/u]$ will give rise to the second $\multimap_{\mathtt{ls}}$-step, since y is the head variable of $y\ r$, which is what is going to be substituted on the head variable of t, *i.e.* $[y/u]$ is an *hereditary* head substitution of t. We use this idea to define the measure. Note that, according to our reasoning, $[y/u]$ is an hereditary head substitution also for $s = (x\ y)[x/(y\ r)[y/u]]$, but we get around these nested cases because we only have to deal with shallow terms.

▶ **Definition 4.8.** *Hereditary head contexts* are generated by the following grammar:

$$HH := H \mid HH[x][x/H].$$

[1] Soundness is Theorem 3 (Section 7) in [16] and completeness follows from the fact that $\lambda\sigma$ enjoys *finite normalization cones*, the abstract rewriting notion studied in that paper.

[2] In particular, it can be shown that $\Lambda_{[\cdot]}$ enjoys pushouts modulo permutation equivalence exactly as the λ-calculus, which implies—as Melliès says in [16]—that its normalization cones are at most singletons (*i.e.* trivial in comparison with those of the $\lambda\sigma$-calculus).

The *head measure* $|t|_{HH}$ of a shallow term t is defined by induction on t:

$$\begin{array}{rlrl} |x|_{HH} &= 0; & |t[x/u]|_{HH} &= |t|_{HH}, \text{ if } t \neq HH[x]; \\ |\lambda x.t|_{HH} &= |t|_{HH}; & |t[x/u]|_{HH} &= |t|_{HH}+1, \text{ if } t = HH[x]; \\ |t\ u|_{HH} &= |t|_{HH}. & & \end{array}$$

Please notice that $|t|_{HH} = 0$ for any λ-term t.

Next lemma proves that $|t|_{HH}$ correctly captures the number of $\multimap_{\mathtt{ls}}$-reductions from t, what is then compactly expressed by the successive corollary. The detailed proof of the lemma is in [4].

▶ **Lemma 4.9** ($|\cdot|_{HH}$ Decreases with $\multimap_{\mathtt{ls}}$). *Let t be a shallow term.*
1. *t is a $\multimap_{\mathtt{ls}}$-normal form iff $|t|_{HH} = 0$.*
2. *$t \multimap_{\mathtt{ls}} u$ implies $|t|_{HH} = |u|_{HH} + 1$.*
3. *$|t|_{HH} > 0$ implies that t is not $\multimap_{\mathtt{ls}}$-normal.*

Proof. 1) By induction on t. 2) By induction on $t \multimap_{\mathtt{ls}} u$. 3) By induction on t. ◀

Summing up, we get:

▶ **Corollary 4.10** (Exact Bound to $\multimap_{\mathtt{ls}}$-sequences). *$t \multimap^n_{\mathtt{ls}} t{\downarrow_\circ}$ iff $n = |t|_{HH}$.*

Proof. By induction on n. For $n = 0$ see Lemma 4.9.1, for $n > 0$ it follows from 4.9.2-3. ◀

Now, we are ready to prove the quadratic relation. The following lemma is the key point for the combinatorial analysis. It shows that if the initial term t of a reduction $\rho : t \multimap^n u$ is a λ-term, then $|u|_{HH}$ is bounded by the number of $\multimap_{\mathtt{dB}}$ steps in ρ.

▶ **Lemma 4.11.** *Let $t \in \mathcal{T}_\lambda$. If $\rho : t \multimap^n u$ then $|u|_{HH} \leq \mathtt{es}(u) = |\rho|_{\mathtt{dB}}$.*

Proof. Note that by definition of $|\cdot|_{HH}$ we get $|u|_{HH} \leq \mathtt{es}(u)$ for any term u. So we only need prove that $\mathtt{es}(u) = |\rho|_{\mathtt{dB}}$. By induction on $k = |\rho|_{\mathtt{dB}}$. If $k = 0$ then ρ is empty, because t is a λ-term and so it is $\multimap_{\mathtt{ls}}$-normal. Then $t = u$ and $\mathtt{es}(u) = 0$. If $k > 0$ then $\rho = \tau; \multimap_{\mathtt{dB}}; \multimap^m_{\mathtt{ls}}$ for some m and some reduction τ. Let r be the end term of τ and s the term s.t. $r \multimap_{\mathtt{dB}} s \multimap^*_{\mathtt{ls}} u$. By *i.h.* $\mathtt{es}(r) = |\tau|_{\mathtt{dB}} = |\rho|_{\mathtt{dB}} - 1$. Now, $\mathtt{es}(s) = \mathtt{es}(r) + 1 = |\rho|_{\mathtt{dB}}$, because each $\multimap_{\mathtt{dB}}$-step creates an explicit substitution. By lemma 3.2 we get that any box subterm of s is a box-subterm of t, and since t is a λ-term, the duplication performed by a $\multimap_{\mathtt{ls}}$-step does not increase the number of explicit substitutions. Therefore, $\mathtt{es}(u) = \mathtt{es}(s) = |\rho|_{\mathtt{dB}}$. ◀

We finally get:

▶ **Theorem 4.12.** *Let $t \in \mathcal{T}_\lambda$. If $\rho : t \multimap^n u$ then $n = O(|\rho|^2_{\mathtt{dB}})$.*

Proof. There exists $k \in \mathbb{N}$ s.t. $\rho = \tau_1; \gamma_1; \ldots; \tau_k; \gamma_k$, where τ_i is a non-empty $\multimap_{\mathtt{dB}}$-reduction and γ_i is a $\multimap_{\mathtt{ls}}$-reduction for $i \in \{1, \ldots, k\}$ and it is non-empty for $i \in \{1, \ldots, k-1\}$. Let $r_1, \ldots, r_k$ be the end terms of $\tau_1, \ldots, \tau_k$, respectively. By Corollary 4.10 $|\gamma_j| \leq |r_j|_{HH}$ and by Lemma 4.11 $|r_j|_{HH} \leq \sum_{i \in \{1,\ldots,j\}} |\tau_i|$. Now $|\rho|_{\mathtt{dB}} = \sum_{i \in \{1,\ldots,k\}} |\tau_i|$ bounds every $|r_j|_{HH}$, hence:

$$\sum_{i \in \{1,\ldots,k\}} |\gamma_i| \leq \sum_{i \in \{1,\ldots,k\}} |r_i|_{HH} \leq k \cdot |\rho|_{\mathtt{dB}}.$$

But k is bounded by $|\rho|_{\mathtt{dB}}$ too, thus $\sum_{i \in \{1,\ldots,k\}} |\gamma_i| \leq |\rho|^2_{\mathtt{dB}}$ and $n \leq |\rho|^2_{\mathtt{dB}} + |\rho|_{\mathtt{dB}} = O(|\rho|^2_{\mathtt{dB}})$. ◀

Putting together the results from the whole of Section 4, we get:

▶ **Corollary 4.13** (Invariance, Part I). *There is a polynomial time algorithm that, given $t \in \mathcal{T}_\lambda$, computes a term u such that $u\!\downarrow = r$ if t has $\rightarrow_{\mathtt{h}}$-normal form r and diverges if u has no $\rightarrow_{\mathtt{h}}$-normal form. Moreover, the algorithm works in polynomial time on the unitary cost of the input term.*

One may now wonder why a result like Corollary 4.13 cannot be generalized to, e.g., leftmost-outermost reduction, which is a normalizing strategy. Actually, linear explicit substitutions *can* be endowed with a notion of reduction by levels capable of simulating the leftmost-outermost strategy in the same sense as linear head-reduction simulates head-reduction here. And, noticeably, the subterm property *continues* to hold. What is not true anymore, however, is the quadratic bound we have proved in this section: in the leftmost-outermost linear strategy, one needs to perform *too many* substitutions not related to any β-redex. If one wants to generalize Corollary 4.13, in other words, one needs to further optimize the substitution process. But this is outside the scope of this paper.

5 $\Lambda_{[\cdot]}$ as an Acceptable Encoding of λ-terms

The results of the last two sections can be summarized as follows: linear explicit substitutions provide both a compact representation for λ-terms and an efficient implementation of β-reduction. Here, efficiency means that the overhead due to substitutions remains under control and is at most polynomial in the unitary cost of the λ-term we start from. But one may wonder whether explicit substitutions are nothing more than a way to *hide* the complexity of the problem under the carpet of compactness: what if we want to get the normal form in the *usual*, *explicit* form? Counterexamples from Section 2, read through the lenses of Theorem 4.13 tell us that that this is *indeed* the case: there are families of λ-terms with polynomial unitary cost but whose normal form intrinsically requires exponential time to be produced.

In this section, we show that this phenomenon is due to the λ-calculus being a very inefficient way to represent λ-terms: even if computing the unfolding of a term $t \in \Lambda_{[\cdot]}$ takes exponential time, *comparing* the unfoldings of two terms $t, u \in \Lambda_{[\cdot]}$ for equality can be done in polynomial time in $|t| + |u|$. This way, linear explicit substitutions are proved to be a succint, acceptable encoding of λ-terms in the sense of Papadimitriou [18]. The algorithm we are going to present is based on dynamic programming: it compares all the *relative unfoldings* of subterms of two terms t and u as above, without really computing those unfoldings. Some complications arise due to the underlying notion of equality for λ-terms, namely α-equivalence, which is coarser than syntactical equivalence. But what is the relative unfolding of a term?

▶ **Definition 5.1** (Relative Unfoldings). The unfolding $t\!\downarrow_C$ of t relative to context C is defined by induction on C:

$$t\!\downarrow_{[\cdot]} := t\!\downarrow; \qquad t\!\downarrow_{u\,C} := t\!\downarrow_C; \qquad t\!\downarrow_{u[x/C]} := t\!\downarrow_C;$$
$$t\!\downarrow_{C\,u} := t\!\downarrow_C; \qquad t\!\downarrow_{\lambda x.C} := t\!\downarrow_C; \qquad t\!\downarrow_{C[x/u]} := t\!\downarrow_C\{x/u\!\downarrow\}.$$

Constraining sets allow to give sensible judgments about the equivalence of terms even when their free variables differ:

▶ **Definition 5.2** (Constraining Sets and Coherence). A *constraining set* A is a set of pairs (x, y) of variable names. Two constraining sets A and B are *coherent* (noted $A \sim B$) if:

Positive Rules

$$\frac{x, y \notin \mathcal{S}}{(x, C) \overset{\{(x,y)\}}{\sim} (y, D)}\ \texttt{var} \qquad \frac{(t, C[[\cdot]\ r]) \overset{v}{\sim} (u, D[[\cdot]\ s]) \qquad (r, C[t\ [\cdot]]) \overset{w}{\sim} (s, D[u\ [\cdot]])}{(t\ r, C) \overset{v \diamond w}{\sim} (u\ s, D)}\ @$$

Error Generation Rules

$$\frac{(t, C[\lambda x.[\cdot]]) \overset{v}{\sim} (u, D[\lambda y.[\cdot]]) \qquad (\exists z \neq y.\{(x, z)\} \in v) \vee (\exists z \neq x.\{(z, y)\} \in v) \vee v = \bot}{(\lambda x.t, C) \overset{\bot}{\sim} (\lambda y.u, D)}\ \lambda_3$$

$$\frac{}{(\lambda x.t, C) \overset{\bot}{\sim} (u\ r, D)}\ \texttt{err}_{\lambda @} \qquad \frac{}{(u\ r, C) \overset{\bot}{\sim} (\lambda x.t, D)}\ \texttt{err}_{@ \lambda} \qquad \frac{x \notin \mathcal{S}}{(x, C) \overset{\bot}{\sim} (u\ r, D)}\ \texttt{err}_{x @}$$

$$\frac{x \notin \mathcal{S}}{(\lambda y.t, C) \overset{\bot}{\sim} (x, D)}\ \texttt{err}_{\lambda x} \qquad \frac{x \notin \mathcal{S}}{(x, C) \overset{\bot}{\sim} (\lambda y.t, D)}\ \texttt{err}_{x \lambda} \qquad \frac{x \notin \mathcal{S}}{(u\ r, C) \overset{\bot}{\sim} (x, D)}\ \texttt{err}_{@ x}$$

Figure 2 Unfolding Rules.

- $(x, y) \in A$ and $(x, z) \in B$ imply $y = z$;
- $(y, x) \in A$ and $(z, x) \in B$ imply $y = z$.

Moreover, A is *auto-coherent* if $A \sim A$. Observe that $\sim$ is not reflexive and that a constraining set is auto-coherent iff it is the graph of a bijection.

As an example $A = \{(x, y), (x, z)\}$ is not auto-coherent, while $B = \{(w, u)\}$ is auto-coherent, like any singleton set of pairs of variables. Moreover, $A \sim B$.

The algorithm we are going to define takes in input a pair (a, b) of terms. We assume them *preprocessed* as follows: the spaces of substituted, abstracted and free names of a and b are all pairwise disjoint and neither a nor b contain any subterm in the form $c[x/d]$, where $x \notin \texttt{fv}(c)$[3]. We also note $\mathcal{S}$ the set of substituted variables of both terms. The whole algorithm is built around the notion of an unfolding judgment:

▶ **Definition 5.3** (Unfolding Judgments and Unfolding Rules). Let $P = (a, b)$ be a preprocessed pair. An *unfolding judgement* is a triple $(t, C), v, (u, D)$, where the *value* v is either $\bot$ or a constraining set A, also noted $(t, C) \overset{v}{\sim} (u, D)$. The rules for deriving unfolding judgments are in Figure 2. An operation $\diamond$ on values is used, which is defined as follows:

1. $v \diamond w = v \cup w$ if $v \neq \bot \neq w$ and $v \sim w$;
2. $v \diamond w = \bot$ if $v \neq \bot \neq w$ and $v \not\sim w$;
3. $v \diamond w = \bot$ if $v = \bot$ or $w = \bot$.

The rules in Figure 2 induce a binary relation $\sqsubset_{a,b}$ on the space of pairs (of pairs) in the form $((t, C), (u, D))$ such that $C[t] = a$ and $D[u] = b$: $(P, Q) \sqsubset_{a,b} (R, S)$ if knowing v such that $P \overset{v}{\sim} Q$ is necessary to compute w such that $R \overset{w}{\sim} S$.

The meaning of unfolding judgements can be summarized as follows. If $(t, C) \overset{A}{\sim} (u, D)$, then $r = t{\downarrow_C}$ and $s = u{\downarrow_D}$ are α-equivalent, provided free variables of r are modified according to A, seen as a substitution (or, analogously, free variables of s are modified according to A^{-1}). If, instead, $(t, C) \overset{\bot}{\sim} (u, D)$, there cannot be any substitution like A above. The rules in Figure 2 are not only a sound but also complete way of checking equality of terms in the just described sense.

[3] Any term can be turned in this form in polynomial time, by $\to_{\text{gc}}$-normalizing it.

Given a preprocessed pair (a, b), checking the equality of $a{\downarrow}$ and $b{\downarrow}$ through a direct use of the formal system above immediately leads to an exponential blowup, which however can be avoided by way of dynamic programming.

An unfolding matrix for a preprocessed pair (a, b) is the data structure storing intemediate judgements produced while deriving a judgement $(a, [\cdot]) \overset{v}{\sim} (b, [\cdot])$. Relying on unfolding matrices ultimately allows to avoid repeating the same computation many times. Formally, an *unfolding matrix* for a preprocessed pair (a, b) is a bidimensional array whose rows are indexed by pairs $(t, C[\cdot])$ such that $C[t] = a$ and whose columns are indexed by pairs $(t, C[\cdot])$ such that $C[t] = b$. Each element of the unfolding matrix can be either a (possibly empty) constraining sets or $\perp$ or $\#$.

Basically, the unfolding checking algorithm simply proceeds by filling an unfolding matrix with the correct values, following the rules in Figure 2 and starting from an unfolding matrix filled with $\#$:

▶ **Definition 5.4** (Unfolding Checking Algorithm)**.** Let $P = (a, b)$ be a preprocessed pair. We define the following algorithm, which will be referred to as the *unfolding checking algorithm*:

1. Initialize all entries in `M` to $\#$;
2. If `M` has no entries filled with $\#$, then go to step 7;
3. Choose (t, C) and (u, D) such that $\mathtt{M}[(t, C)][(u, D)] = \#$, and such that $\mathtt{M}[P_1][Q_1], \ldots, \mathtt{M}[P_n][Q_n]$ are all different from $\#$, where $(P_1, Q_1), \ldots, (P_n, Q_n)$ are the immediate predecessors of $((t, C), (u, D))$ in $\sqsubset_{a,b}$;
4. Compute v such that $(t, C) \overset{v}{\sim} (u, D)$;
5. $\mathtt{M}[(t, C)][(u, D)] \leftarrow v$;
6. Go to step 2;
7. Return `yes` if $\mathtt{M}[(a, [\cdot])][(b, [\cdot])]$ is a constraining set which is the identity, otherwise return `no`.

The Unfolding Checking Algorithm has a simple structure: initially `M` only contains $\#$, and steps $2 - 5$ are repeated until `M` does not contain any $\#$. At each iteration, a cell containing $\#$ is assigned a value v, while the rest of `M` is left unchanged.

It is now time to prove that the Unfolding Checking Algorithm is correct, i.e., that it gives correct results, if any:

▶ **Theorem 5.5** (Correctness)**.** *The Unfolding Checking Algorithm, on input (a, b), returns* `yes` *iff $a{\downarrow} = b{\downarrow}$.*

Proof. This can be done by proving the following two lemmas:

- If $(t, C) \overset{v}{\sim} (u, D)$ and $v \neq \perp$ then v induces two renamings σ and θ such that $t{\downarrow_C}\sigma = u{\downarrow_D}$ and $u{\downarrow_D}\theta = t{\downarrow_C}$;
- If $(t, C) \overset{\perp}{\sim} (u, D)$, then there is no such pair of renamings for $t{\downarrow_C}$ and $u{\downarrow_D}$.

Both can be proved by induction on the derivation of $(t, C) \overset{A}{\sim} (u, D)$. For details, see [4]. ◀

This is not the end of the story, however — one also needs to be sure about the time complexity of the algorithm, which turns out to be polynomial:

▶ **Proposition 5.6** (Complexity)**.** *The Unfolding Checking Algorithm works in time polynomial in $|a| + |b|$.*

Proof. The following observations are sufficient to obtain the thesis:

- The number of entries in `M` is $|a||b|$ in total.

- At every iteration, one element of `M` changes its value from $\#$ to some non-blank v.
- Step 2 can clearly be performed in time polynomial in $|a| + |b|$.
- Computing the predecessors of a pair P can be done in polynomial time, and so Step 3 can itself be performed in time polynomial in $|a| + |b|$.
- Rules in Figure 2 can all be applied in polynomial time, in particular because the cardinality of any constraining set produced by the algorithm can be $|a| + |b|$, at most. As a consequence, Step 4 can be performed in polynomial time.

This concludes the proof. ◀

6 Encoding Turing Machines

A cost model *for computation time* is said to be invariant if it is polynomially related to the standard cost model on Turing machines. In sections 3 and 4, we proved that head reduction of any λ-term t can be performed on a Turing machine in time polynomial in the number of β-steps leading t to its normal form (provided it exists). This is proved through explicit substitutions, which in Section 5 are shown to be a reasonable representation for λ-terms: two terms t and u in $\Lambda_{[\cdot]}$ can be checked to have α-equivalent unfoldings in polynomial time.

The last side of the story is still missing, however. In this section, we will show how head reduction can simulate Turing machine computation in such a way that the unitary cost of the simulating λ-term is polynomially related to the running time of the encoded Turing machine. Results similar to the one we are going to present are common for the λ-calculus, so we will not give all the details, which can anyway be found in [4].

The simplest objects we need to encode in the λ-calculus are finite sets. Elements of any finite set $A = \{a_1, \ldots, a_n\}$ can be encoded as follows: $\lceil a_i \rceil^A \equiv \lambda x_1. \ldots .\lambda x_n.x_i$. Notice that the above encoding induces a total order on A such that $a_i \leq a_j$ iff $i \leq j$. Other useful objects are finite strings over an arbitrary alphabet, which will be encoded using a scheme attributed to Dana Scott. Let $\Sigma = \{a_1, \ldots, a_n\}$ be a finite alphabet. A string in $s \in \Sigma^*$ can be represented by a value $\lceil s \rceil^{\Sigma^*}$ as follows, by induction on the structure of s:

$$\lceil \varepsilon \rceil^{\Sigma^*} \equiv \lambda x_1. \ldots .\lambda x_n.\lambda y.y; \qquad\qquad \lceil a_i r \rceil^{\Sigma^*} \equiv \lambda x_1. \ldots .\lambda x_n.\lambda y.x_i \lceil r \rceil^{\Sigma^*}.$$

Observe that representations of symbols in Σ and strings in Σ^* depend on the cardinality of Σ. In other words, if $s \in \Sigma^*$ and $\Sigma \subset \Delta$, $\lceil s \rceil^{\Sigma^*} \neq \lceil s \rceil^{\Delta^*}$.

Of course, one should be able to very easily compute the encoding of a string obtained by concatenating another string with a character. Moreover, the way strings are encoded depends on the underlying alphabet, and as a consequence, we also need to be able to convert representations for strings in one alphabet to corresponding representations in another, different, alphabet. This can be done efficiently in the λ-calculus by way of a term $AC(\Sigma)$ which append a character to a string (both expressed in the alphabet Σ) and a term $CS(\Sigma, \Delta)$ which converts a string $s \in \Sigma^*$ into another string in Δ^* obtained by replacing any character in $\Sigma - \Delta$ by the empty string. $AC(\Sigma)$ works in time independent on the size of the input, while $CS(\Sigma, \Delta)$ works in time *proportional* to the size of the argument.

A configuration (u, a, v, q) of a Turing machine $\mathcal{M} = (\Sigma, a_{blank}, Q, q_{initial}, q_{final}, \delta)$ consists of the portion of the tape on the left of the head (namely $u \in \Sigma^*$), the symbol under the head ($a \in \Sigma$), the tape on the right of the head ($v \in \Sigma^*$), and the current state $q \in Q$. Such a configuration is represented by the term $\lceil (u, a, v, q) \rceil^{\mathcal{M}} \equiv \lambda x.x \lceil u^r \rceil^{\Sigma^*} \lceil a \rceil^{\Sigma} \lceil v \rceil^{\Sigma^*} \lceil q \rceil^{Q}$.

We now encode a Turing machine $\mathcal{M} = (\Sigma, a_{blank}, Q, q_{initial}, q_{final}, \delta)$ in the λ-calculus. Suppose $\Sigma = \{a_1, \ldots, a_{|\Sigma|}\}$ and $Q = \{q_1, \ldots, q_{|Q|}\}$. The encoding of $\mathcal{M}$ is defined around three λ-terms:

- First of all, we need to be able to build the initial configuration for u from u itself. This can be done in time proportional to $|u|$ by a term $\mathcal{I}(\mathcal{M},\Delta)$, where Δ is the alphabet of u, which can be different from Σ. $\mathcal{I}(\mathcal{M},\Delta)$ simply converts u into the appropriate format by way of $CS(\Delta,\Sigma)$, and then packages it into a configuration.
- Then, we need to extract a string from a final configuration C for the string. This can be done in time proportional to the size of C by a term $\mathcal{F}(\mathcal{M},\Delta)$, which makes essential use of $CS(\Sigma,\Delta)$.
- Most importantly, we need to be able to simulate the transition function of $\mathcal{M}$, i.e. compute a final configuration from an initial configuration (if it exists). This can be done with cost proportional to the number of steps $\mathcal{M}$ takes on the input, by way of a term $\mathcal{T}(\mathcal{M})$. The term $\mathcal{T}(\mathcal{M})$ just performs case analysis depending on the four components of the input configuration, then manipulating them making use of $AC(\Sigma)$.

A peculiarity of the proposed encoding of Turing machines is the use of Scott numerals (instead of Church numerals), which make the simulation work even when head reduction is the underlying notion of computation. Another crucial aspect is *continuation passing*: the λ-terms cited above all take an additional parameter to which the result is applied after being computed. At this point, we can give the desired simulation result:

▶ **Theorem 6.1** (Invariance, Part II). *If $f : \Delta^* \to \Delta^*$ is computed by a Turing machine $\mathcal{M}$ in time g, then there is a term $\mathcal{U}(\mathcal{M},\Delta)$ such that for every $u \in \Delta^*$, $\mathcal{U}(\mathcal{M},\Delta)\lceil u\rceil^{\Delta^*} \to_{\mathtt{h}}^{n} \lceil f(u)\rceil^{\Delta^*}$ where $n = \mathcal{O}(g(|u|) + |u|)$.*

Proof. Simply define $\mathcal{U}(\mathcal{M},\Delta) \equiv \lambda u.\mathcal{I}(\mathcal{M},\Delta)(\lambda x.\mathcal{T}(\mathcal{M})(\lambda y.\mathcal{F}(\mathcal{M},\Delta)(\lambda w.w)y)x)u$. It is routine to prove the thesis. ◀

Noticeably, the just described simulation induces a linear overhead: every step of $\mathcal{M}$ corresponds to a constant cost in the simulation, the constant cost not depending on the input but only on $\mathcal{M}$ itself.

7 Conclusions

The main result of this paper is the first invariance result for the λ-calculus when reduction is allowed to take place in the scope of abstractions. The key tool to achieve invariance are linear explicit substitutions, which are *compact* but *manageable* representations of λ-terms.

Of course, the main open problem in the area, namely invariance of the unitary cost model for any normalizing strategy (e.g. for the strategy which always reduces the leftmost-outermost redex) remains open. Although linear explicit substitutions cannot be *directly* applied to this problem, the authors strongly believe that this is anyway a promising direction, on which they are actively working at the time of writing.

References

1 M. Abadi, L. Cardelli, P. L. Curien, and J. J. Levy. Explicit substitutions. *Journal of Functional Programming*, 1:31–46, 1991.

2 B. Accattoli. An abstract factorization theorem for explicit substitutions. Accepted at RTA 2012.

3 B. Accattoli. *Jumping around the box: graphical and operational studies on Lambda Calculus and Linear Logic.* Ph.D. Thesis, Università di Roma La Sapienza, 2011.

4 B. Accattoli and U. Dal Lago. On the invariance of the unitary cost model for head reduction (long version). Available at `http://arxiv.org/abs/1202.1641`.

5 B. Accattoli and S. Guerrini. Jumping boxes. In *Proceedings of CSL 2009*, pages 55–70, 2009.

6 B. Accattoli and D. Kesner. The structural λ-calculus. In *Proceedings of CSL 2010*, volume 6247 of *LNCS*, pages 381–395. Springer, 2010.

7 B. Accattoli and D. Kesner. The permutative -calculus, 2012.

8 M. Avanzini and G. Moser. Complexity analysis by graph rewriting. In *Proceedings of FLOPS 2010*, volume 6009 of *LNCS*, pages 257–271. Springer, 2010.

9 Martin Avanzini and Georg Moser. Closing the gap between runtime complexity and polytime computability. In *Proceedings of RTA 2010*, volume 6 of *LIPIcs*, pages 33–48. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

10 U. Dal Lago and S. Martini. The weak lambda-calculus as a reasonable machine. *Theoretical Computer Science*, 398:32–50, 2008.

11 U. Dal Lago and S. Martini. On constructor rewrite systems and the lambda-calculus. In *Proceedings of ICALP 2009*, volume 5556 of *LNCS*, pages 163–174. Springer, 2009.

12 U. Dal Lago and S. Martini. Derivational complexity is an invariant cost model. In *Proceedings of FOPARA 2010*, volume 6324 of *LNCS*, pages 88–101. Springer, 2010.

13 V. Danos and L. Regnier. Head linear reduction. Submitted. Available at `http://iml.univ-mrs.fr/~regnier/articles/pam.ps.gz`, 2004.

14 G. Mascari and M. Pedicini. Head linear reduction and pure proof net extraction. *Theoretical Computer Science*, 135(1):111–137, 1994.

15 P.-A. Melliès. Typed lambda-calculi with explicit substitutions may not terminate. In *Proceedings of TLCA 1995*, volume 902 of *LNCS*, pages 328–334. Springer, 1995.

16 P.-A. Melliès. Axiomatic rewriting theory II: the $\lambda\sigma$-calculus enjoys finite normalisation cones. *Journal of Logic and Computation*, 10(3):461–487, 2000.

17 R. Milner. Local bigraphs and confluence: Two conjectures: (extended abstract). *ENTCS*, 175(3):65–73, 2007.

18 C. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

19 G. D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science*, 1(2):125–159, 1975.

20 L. Regnier. *Lambda-calcul et réseaux*. Thèse de doctorat, Univ. Paris VII, 1992.

21 D. Sands, J. Gustavsson, and A. Moran. Lambda calculi and linear speedups. In *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, number 2566 in LNCS, pages 60–82. Springer, 2002.

22 K. Terui. Light affine lambda calculus and polynomial time strong normalization. *Archive for Mathematical Logic*, 46(3-4):253–280, 2007.

23 P. van Emde Boas. Machine models and simulation. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 1–66. MIT Press, 1990.

A Term Rewriting System for Kuratowski's Closure-Complement Problem*

Osama Al-Hassani[1], Quratul-ain Mahesar[1], Claudio Sacerdoti Coen[2], and Volker Sorge[1]

1 School of Computer Science, University of Birmingham
Edgbaston, United Kingdom
O.Al-Hassani@cs.bham.ac.uk,Q.Mahesar@cs.bham.ac.uk,V.Sorge@cs.bham.ac.uk

2 Dipartimento di Scienze dell'Informazione, Università di Bologna
Mura Anteo Zamboni, 7 (BO) Italy
sacerdot@cs.unibo.it

Abstract

We present a term rewriting system to solve a class of open problems that are generalisations of Kuratowski's closure-complement theorem. The problems are concerned with finding the number of distinct sets that can be obtained by applying combinations of axiomatically defined set operators. While the original problem considers only closure and complement of a topological space as operators, it can be generalised by adding operators and varying axiomatisation. We model these axioms as rewrite rules and construct a rewriting system that allows us to close some so far open variants of Kuratowski's problem by analysing several million inference steps on a typical personal computer.

1998 ACM Subject Classification G.2.1 Combinatorics

Keywords and phrases Kuratowski's closure-complement problem, Rewriting system

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.38

Category Regular Research Paper

1 Introduction

In 1922 Kuratowski asked and solved the following question on an arbitrary topological space: how many different combinations of the operators of complement and closure exist? The number turns out to be just 14 and the proof is quite small. The problem has been generalised in many different ways to consider other operators, such as union or intersection, or slightly different settings, such as point free topology (locale theory). The solution to a generalised version can be a significantly larger number of combinations, but it could also be a proof that infinitely many combinations exist.

Computing finite large solutions, or obtaining an intuition for infinite variants is unfeasible by hand and therefore computer automation is crucial to our solutions of the problems. Solutions or partial solutions can be represented as directed graphs whose vertices are equivalence classes of provably equal combinations of operators and whose arcs represent the order relation. Graphs can be constructed iteratively by systematically adding more combinations of iterators, merging distinct vertices once it can be shown that they contain

* This work was partially supported by the Italian PRIN project McTafi (Metodi Costruttivi in Topologia, Algebra e Fondamenti dell'Informatica.)

$$\frac{}{x \le x}(\texttt{reflexive}) \qquad \frac{x \le y \quad y \le z}{x \le z}(\texttt{transitive}) \qquad \frac{x \le y \quad y \le x}{x = y}(\texttt{antisymmetric})$$

$$\frac{x \le y}{-y \le -x}(\texttt{antimonotone}) \qquad \frac{}{x \le --x}(\texttt{saturates}) \qquad \frac{}{---x = -x}(\texttt{quasi-idempotent})$$

$$\frac{}{i(x) \le x}(\texttt{reduces}) \qquad \frac{x \le y}{i(x) \le i(y)}(\texttt{monotone}) \qquad \frac{}{i(x) = i(i(x))}(\texttt{idempotent})$$

$$\frac{}{x \le c(x)}(\texttt{saturates}) \qquad \frac{x \le y}{c(x) \le c(y)}(\texttt{monotone}) \qquad \frac{}{c(x) = c(c(x))}(\texttt{idempotent})$$

$$\frac{}{c(-x) \le -i(x)}(\texttt{compatible-1}) \qquad \frac{}{i(-x) \le -c(x)}(\texttt{compatible-2})$$

Figure 1 The inference rules that define the problem.

provably equal combinations. For problems with finite solutions, this process terminates once a fixpoint is reached, in the sense that new vertices are not added, or vertices will never be merged in the future. The resulting graph then represents all possible distinct combinations. In a case where a solution is infinite, there exist infinitely many distinct combinations and finite graphs can only ever represent an approximation of the solution. Nevertheless these approximations can offer crucial information by exhibiting regularities in the growth of the graph, to suggest the existence of an infinite subgraph of distinct vertices that will never become equal, thereby leading to the essential proof idea.

We present a generalised Kuratowski problem (§2) — originally proposed by Sambin in [7], Section 2.4, and still listed as an open problem in [8] — which is particularly demanding in terms of size of the approximating graphs. Indeed, in order to exhibit sufficient evidence of regularities in the graph we need to compute several million edges, i.e., we need to prove several million lemmas on relations between pairs of operator combinations. Since traditional automated theorem proving techniques are unsuitable for this task, we have developed a term rewriting system that models all inference rules of the problem in a uniform way and, coupled with a particular strategy and some standard graph algorithms, can show the large numbers of necessary lemmas in a few minutes (§3). We then demonstrate how the rewriting system can be further enhanced, whilst retaining both correctness and completeness by exploiting regularities in the partial solutions of the problem (§4). The implementation of our ideas have not only enabled us to prove the infinite nature of the generalised Kuratowski problem, which was up to now unknown (§6), but also serves as a basis to tackle other variants of Kuratowski's problem (§5).

2 The Problem

Kuratowski's classical closure-complement problem [5] can be solved by observing that the following identities hold for the interior operator i, the closure operator c, and the complement '$-$' for subsets x of an arbitrary topological space. (i) $c(c(x)) = c(x)$, (ii) $--x = x$, (iii) $i(x) = -c(-x)$, and (iv) $c(i(c(i(x)))) = c(i(x))$. Applying the previous equalities only, one can show that there exist at most 14 distinct subsets one can obtain by different combinations of the three operators. As a consequence of equation (iii), all combinations can be expressed in terms of closure and complement, only. Furthermore, one can indeed construct a model, that is, a space in which this upper bound can be achieved.

The observation that the previous identities are sufficient to solve the problem allows to restate Kuratowski's classical problem in point-free topology. It is sufficient to forget the concrete definition of the operators and to define them axiomatically by means of the four identities above. The domain of the operators is thus no longer required to be the power set of a topological space, but it can be any arbitrary set.

In the rest of the paper we are interested in the generalisation of the point-free version of Kuratowski's problem introduced by Sambin in [7], Section 2.4. Corsi's master thesis [3] studied the problem under the supervision of Sambin, but the problem remained open and only minimal progress towards a solution was achieved. The generalisation is obtained by introducing a partial order relation $\leq$ — that captures the inclusion relation for subsets — and relaxing the axioms for the operators as given in Figure 1 in a rule format. Observe that the relaxed axiomatisation, effectively turns i into a reduction operator, c into a saturation operator, and $-$ into a pseudo-complement. Observe also that the two compatibility requirements are reminiscent of the classical equation $i(x) = -c(-x)$ (dually $c(x) = -i(-x)$). Indeed, if we define $i(x)$ as $-c(-x)$ and we also assume $--x = x$ for all x, then both compatibility axioms can be derived.

An example model for the axioms can be obtained by combining the definitions of interior, closure and complement with the rules of intuitionistic logic. Given a topological space $(P, \mathcal{O})$, the interior of a set x is defined as $\{\alpha \mid \exists y \in \mathcal{O},\ \alpha \in y \wedge \forall \beta \in y.\beta \in x\}$, and a definition of the closure of x that avoids any reference to negation is $\{\alpha \mid \forall y \in \mathcal{O},\ \alpha \in y \Rightarrow \exists \beta \in y.\beta \in x\}$ (the set of all accumulation points of x). Thus the notion of interior hides an $\exists\forall$ combination and that of closure a $\forall\exists$. The complement of x is $\{\alpha \mid \neg(\alpha \in x)\}$ and it hides a negation. Finally, the subset relation (axiomatised as $\leq$) hides implication: $x \subseteq y$ iff $\forall \alpha,\ \alpha \in x \Rightarrow \alpha \in y$. All the axioms presented are thus simply obtained from the properties of negation and the quantifiers in intuitionistic logic. For instance, from the intuitionistic principle $A \Rightarrow \neg\neg A$ we obtain $x \leq --x$ and from the DeMorgan laws for quantifiers we obtain the two compatibility relations: For example, $\forall\exists\neg \Rightarrow \neg\exists\forall$ becomes $c(-x) \leq -i(x)$.

Since we are effectively interested in the number of different combinations of operators that can lead to distinct sets when applied to any subset of a topological space, we define the generalised Kuratowski problem in terms of equivalent operator combinations.

▶ **Definition 1** (Generalised Kuratowski closure-complement problem)**.** Let $(P, \leq)$ be any partially ordered set and let $\{i, c, -\}$ be the set of operators on P axiomatised as in Figure 1. Let $S = \{i, c, -\}^*$ be the set of all words over the operators (i.e., all possible finite combinations). We define the order relation $\leq$ on S for all $w_1, w_2 \in S$ by: $w_1 \leq w_2$ iff $w_1(x) \leq w_2(x)$ for all $x \in P$. Finally, let $\equiv$ over S be the symmetric closure of $\leq$. The generalised Kuratowski closure-complement problem then consists in computing the cardinality of $S_{/\equiv}$, the set of equivalence classes of S modulo $\equiv$.

Useful for the remainder of our considerations is to define the canonical representative of an equivalence class $[w]_{/\equiv} \in S_{/\equiv}$ as the minimum element of the set according to the shortlex order[1]. Furthermore, we can naturally extend the relation $\leq$ on S to equivalence classes.

Since the cardinality of $S_{/\equiv}$ is not necessarily finite, for practical purposes it is necessary to define finite approximations to the solution.

▶ **Definition 2** (n^{th} approximation)**.** Let $S_n = \{i, c, -\}^{\leq n} \subset S$ be the set of all operator combinations up to order n. For $w_1, w_2 \in S_n$ we define $\leq_n$ as $w_1 \leq_n w_2$ iff for all $x \in P$ we

[1] Two words are in the shortlex order relation when the first is shorter or, in case they have the same length, when the first comes first in lexicographical order.

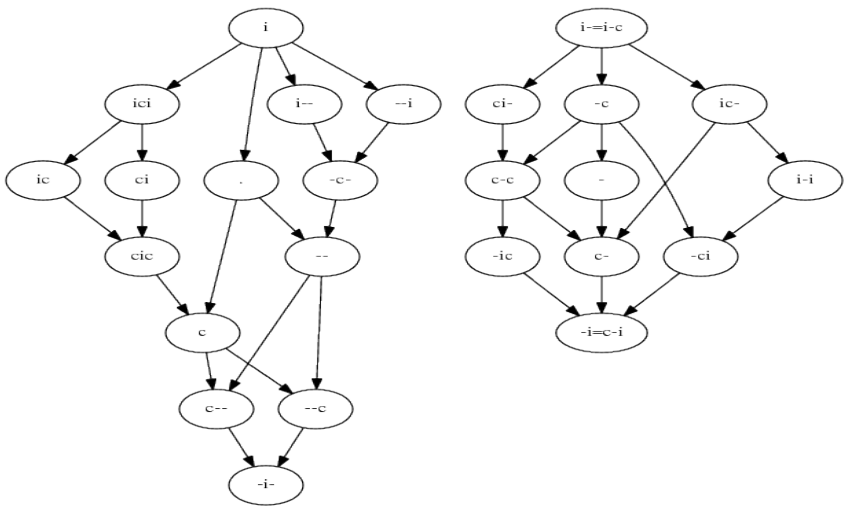


Figure 2 Approximating graph of order 3 for the generalised problem.

can derive $w_1(x) \leq w_2(x)$ by applying the axioms from Figure 1 to elements $w \in S_n$ only (i.e., we restrict derivations to combinations of maximally n operators).

Finally, let $\equiv_n$ the symmetric closure of $\leq_n$. Then the n^{th} approximation of the generalised Kuratowski closure-complement problem is defined as computing the cardinality of $S_{n/\equiv_n}$.

The n^{th} approximation of the problem can be visually represented as a directed graph whose vertices are the equivalence classes of $S_{n/\equiv_n}$ and whose edges represent one step of the $\leq_n$ relation.

▶ **Definition 3** (Approximating graph of order n)**.** Let $G = (V, A)$ be a directed graph, where we define the set of vertices $V = S_n$ and the set of arcs A by $(v_1, v_2) \in A$ iff $v_1 \leq_n v_2$ for $v_1, v_2 \in V$.

Now let V' be the set of all strongly connected components in G. We then define the *approximating graph of order* n as $G' = (V', A')$ where $(v'_1, v'_2) \in A'$ iff $v'_1 \leq_n v'_2$ for $v'_1, v'_2 \in V'$.

Note that the approximating graph can be represented in transitively reduced form, exploiting the transitivity of the $\leq_n$ relation. We also observe that every vertex in the approximating graph contains all the elements of the equivalence class it represents. Thus the graph itself provides a solution to the n^{th} approximation problem as the number of vertices in the graph is the cardinality of $S_{n/\equiv_n}$. Consequently, our goal is effectively to construct the graph by partitioning S_n into equivalence classes, which amounts to an inference procedure that determines if $[w_1]_{/\equiv} \leq_n [w_2]_{/\equiv}$ for $[w_1]_{/\equiv_n}, [w_2]_{/\equiv_n} \in S_n/ \equiv_n$.

Figure 2 shows the approximating graph of order 3 for the generalised Kuratowski closure-complement problem. The vertices are subsets of S_3, where only three vertices represent equivalence classes with more than one element. Note that '.' corresponds to the empty word ϵ.

We note that the n^{th} approximation is an approximation to the original problem in two ways. First of all it only shows classes whose canonical representative has length at most n. More importantly, however, it does not grant that two distinct classes in the n^{th} approximation will remain distinct for every $(n+m)^{\text{th}}$ approximation. Thus the cardinality of the graph may decrease or increase when moving to larger values of n. Nevertheless, the approximation procedure is monotone in the following sense: if two words belong to the same class in the n^{th} approximation, they will belong to the same class in any $(n+m)^{\text{th}}$ approximation. Moving to the $(n+1)^{\text{th}}$ approximation can only collapse more classes or create new ones made only of words of length $n+1$.

The following theorem holds.

▶ **Theorem 4.** *If the solution of the generalised problem is finite, then there exists an n such that every $(n+m)^{th}$ approximation is isomorphic (as a directed acyclic graph) to the solution.*

The theorem says that approximations *stabilise*, in the sense that larger approximations only augment the cardinality of the equivalence classes, but they do not collapse any existent distinct classes, nor do they add new arcs to the approximating graph.

The theorem does not provide an effective way to decide if an approximation is (isomorphic to) the solution. We postulate the following conjecture that would provide a simple decision procedure to recognise solutions.

▶ **Conjecture 5.** *There exists an m such that, if for a given n the $n^{th}$and the $(n+m)^{th}$approximations are isomorphic, then they are isomorphic to the solution.*

We have not tried to prove the conjecture yet and the proof does not seem to be simple. In particular, we do not know what is the m for the set of axioms considered. Nevertheless, we can employ an alternative to the conjecture to recognise which approximations are solutions. Let us assume that at a certain point the approximations seem to stabilise, i.e., the $(n+1)^{\text{th}}$ approximation is equal to the n^{th} approximation. We can build a *syntactic model* of the solution as follow. We take the set P of all strings w made from $\{i, c, -\}$ such that w is a canonical representative of an equivalence class in the n^{th} approximation. Then we define an $\leq$ relation over P by taking the $\leq_n$ relation. The i, c and $-$ operators are obtained as finite maps that associate to each $w \in P$ the canonical representative of $i \circ w$ (respective $c \circ w$ and $- \circ w$) in the n^{th} approximation. In order to verify if $(P, i, c, -)$ is a model for the problem, we can use a computer algebra system or a theorem prover to verify that all axioms hold. If they do, the n^{th} approximation is isomorphic to the solution of the generalised problem because the model shows that all classes are distinct and moreover the number of classes is maximal because we have only equated combinations that had to be equated because proved to be equal. Otherwise, we start computing larger approximations and we will eventually find an $(n+m)^{\text{th}}$ approximation that is not isomorphic to the n^{th} approximation that, a posteriori, was not stable.

A priori, if the conjecture is false it may be that all syntactic models built from approximations that seem to be stable (i.e. $(n+1)^{\text{th}}$ isomorphic to n^{th}) turn out to be wrong. However, as we will see in the conclusions, this has not been the case for the different instances of the generalised problem that we considered and that seemed to be stable.

The following theorem, instead, is obvious:

▶ **Theorem 6.** *If the solution of the generalised problem is infinite, then there exists an infinite increasing sequence of approximations with larger and larger cardinalities.*

Our experience shows that in this case a clear pattern emerges, which after some time allows us to predict what new classes will be generated passing from any n^{th} approximation to the $(n+1)^{\text{th}}$ approximation. This prediction can then be manually turned into a proof that these new classes will never be collapsed in later approximations and therefore the solution is infinite.

Consequently, in the rest of the paper we will focus on finding a solution to the n^{th} approximation of the problem.

3 The Basic Rewriting System

We now present a method to compute approximating graphs by constructing a rewriting system directly from the axioms given in Figure 1. We present the term rewriting system in a standard way employing terminology used in standard references such as [1]. Nevertheless, the system can also be understood as an instance of generalised equational reasoning as defined in [6], which corresponds more to the actual form in which the system was developed.

A preliminary observation is the fact that in the axiomatisation of the problem we can replace the equality with a $\leq$ in all three idempotency inference rules as the (anti)monotonicity of the respective operator yields the equality automatically. For instance, idempotency of the i operator can be replaced by the axiom $i(x) \leq i(i(x))$ since, by monotonicity, we already have $i(i(x)) \leq i(x)$. Therefore, the only rule that employs an equality remains the antisymmetry rule for $\leq$.

The approximating graph of order n can now be computed in two steps. In the first step we compute the initial directed graph G from Definition 3 whose vertices are all the elements of S_n (words of length at most n) and whose arcs represent all pairs $(w_1, w_2) \in \leq_n$. The anti-symmetric rule of the $\leq$ relation and, more generally, equalities are not used in this step. In the second step we apply a standard connected component algorithm to this graph. Since a connected component is made of all vertices that are mutually reachable, i.e., mutually less or equal, by antisymmetry of $\leq$ they are all equal. The resulting graph is then the approximating graph of order n we are looking for.

Since the second step is completely standard and we can employ implementations out-of-the-box, we will only focus on developing the first step here.

In order to compute the first directed graph $G = (V, A)$ it is sufficient to find all pairs of vertices $(w_1, w_2) \in A$ in the transitive reduction of the graph of $\leq_n$ (recall that following Definition 3 w_1, w_2 are words of length $\leq n$ in S_n), since the connected components algorithm does not distinguish between a transitively reduced and a transitively closed graph. In other words, we are looking for all pairs $(w_1, w_2) \in A$ such that $w_1 \leq_n w_2$ and there is no $w_3 \in V$ such that $w_1 \leq_n w_3$ and $w_3 \leq_n w_2$.

By a close inspection of the rules that have a premise, it is easy to notice that all applications of the transitive rules can be pushed towards the root of the derivation tree. For instance, consider the monotone rule for i and assume (by induction hypothesis) that the derivation of the premise $x \leq y$ is obtained by means of a transitive rule whose premises are $x \leq z$ and $z \leq y$. It is therefore possible to conclude both $i(x) \leq i(z)$ and $i(z) \leq i(y)$ and then, with one final application of transitivity, that $i(x) \leq i(z)$. Thus, since we are interested only in the transitively reduced graph, we can also avoid the use of the transitive and reflexive properties of $\leq$.

The final preliminary observation is that, to compute all pairs (w_1, w_2) in the transitively reduced graph G it is sufficient for every word $w \in S_n$ to compute the two sets $w\!\downarrow = \{w' \mid w' \leq_n w \wedge |w'| \leq |w|\}$ and $w\!\uparrow = \{w' \mid w \leq_n w' \wedge |w'| \leq |w|\}$ where $|.|$ is the length of the two combinations. The final set is just given by

$$\bigcup_{w \in S_n} (\{(w, w') \mid w' \in w\!\uparrow\} \cup \{(w', w) \mid w' \in w\!\downarrow\})$$

In order to compute $w\!\downarrow$ and $w\!\uparrow$ we introduce the non confluent, noetherian term rewriting system presented in Figure 3. The term rewriting system manipulates both active configurations of the form $\langle w_1, w_2, d\rangle$ (where $d \in \{\leq, \geq\}$) and stuck terms w. The intended big step semantics of the rewriting system is the following: an initial term $\langle \epsilon, w, d\rangle \rhd^* w'$ iff

(`saturates`)	(`antimonotone`)	(`quasi-idempotent`)
$\langle w_1, --w_2, \geq\rangle \rhd w_1w_2$	$\langle w_1, -w_2, d\rangle \rhd \langle w_1-, w_2, d^{-1}\rangle$	$\langle w_1, ---w_2, \geq\rangle \rhd w_1-w_2$
(`reduces`)	(`monotone`)	(`idempotent`)
$\langle w_1, iw_2, \leq\rangle \rhd w_1w_2$	$\langle w_1, iw_2, d\rangle \rhd \langle w_1i, w_2, d\rangle$	$\langle w_1, iiw_2, \geq\rangle \rhd w_1iw_2$
(`saturates`)	(`monotone`)	(`idempotent`)
$\langle w_1, cw_2, \geq\rangle \rhd w_1w_2$	$\langle w_1, cw_2, d\rangle \rhd \langle w_1c, w_2, d\rangle$	$\langle w_1, ccw_2, \leq\rangle \rhd w_1cw_2$
(`compatible-1`)	(`compatible-2`)	
$\langle w_1, c-w_2, \leq\rangle \rhd w_1-iw_2$	$\langle w_1, i-w_2, \leq\rangle \rhd w_1-cw_2$	

Figure 3 The non confluent, noetherian term rewriting system to compute $w{\downarrow}$ and $w{\uparrow}$.

wdw' and $|w'| \leq |w|$. In particular, $w{\downarrow}$ can be computed as $\{w' \mid \langle\epsilon, w, \geq\rangle \rhd^* w'\}$ and $w{\uparrow}$ as $\{w' \mid \langle\epsilon, w, \leq\rangle \rhd^* w'\}$.

The small step semantics of the rewriting system is more technical and it involves generic configurations $\langle w_1, w_2, d\rangle$. The idea is that an initial reduction trace $\langle\epsilon, w, d\rangle \rhd^n \langle w_1, w_2, d'\rangle$ represents a partial derivation of wdw' for some yet unknown w'. Two invariants say that $|w_1| = n$ and $w = w_1w_2$. The partial derivation built in a top-down manner starts with exactly n monotonicity/anti-monotonicity rules: if $w_1 = o_1 \ldots o_n$ where $o_j \in \{-, i, c\}$ then the j-th inference rule in the partial derivation is the monotonicity/anti-monotonicity rule for o_j. Moreover, the hypothesis of the partial derivation is $w_2d'w_2'$ for some yet unknown w_2' such that $w' = w_1w_2'$. According to this interpretation, a reduction trace $\langle\epsilon, w, d\rangle \rhd^* \langle w_1, w_2, d'\rangle \rhd w'$ corresponds to a derivation of wdw' where there is a w_2' such that $w' = w_1w_2'$, the last inference rule in the top-down construction is an axiom that proves $w_2d'w_2'$ and $|w_2'| \leq |w_2|$. The proof that reduction traces of length n correspond to partial derivation trees of height n having the property just described is by induction on n. We only sketch here one case of the proof.

Each rule in Figure 3 corresponds to the rule with the same name in Figure 1. It means that applying the reduction rule adds the corresponding inference rule to the partial proof tree. The most interesting rule is the rule `antimonotone`: In order to proceed in the derivation we use one more application of antimonotonicity of complement by pushing $-$ on top of the stack w_1 and looking for a new derivation for $w_2d^{-1}w'$. To see that the rule is correct, assume that $\langle\epsilon, w, d\rangle \rhd^n \langle w_1, -w_2, d'\rangle \rhd \langle w_1-, w_2, d'^{-1}\rangle$. By induction hypothesis there is a partial proof derivation of wdw' built top-down that starts with monotone/anti-monotone rules for the operators in w_1 and whose hypothesis is $-w_2d'w''$ for some yet unknown w'' such that $w' = w_1w''$. By applying anti-monotonicity of $-$ we obtain a new partial proof derivation of wdw' whose new hypothesis is $w_2d'^{-1}w'''$ and such that $w' = w_1w'' = w_1-w'''$. The reduction rule is therefore correct and by applying it we discover that $w'' = -w'''$ or, equivalently, that the next rule in the combination w' after w_1 is $-$.

Strong normalisation of the term rewriting system can simply be proved by induction on the length of the second component of active configurations, which always decreases by one in all (anti)monotonicity rules. All remaining rules produce a stuck term.

By inspection of all the rules, it is easy to prove (by induction on the second component of an active configuration) that if $\langle\epsilon, w, d\rangle \rhd^* w'$ then $|w'| \leq |w|$. Moreover, if $\langle\epsilon, w, d\rangle \rhd^* w'$ and $|w'| = |w|$ then $\langle\epsilon, w', d^{-1}\rangle \not\rhd^* w$. This is important for efficiency reasons since it means that we are never generating the same arc twice (as w_1dw_2 and $w_2d^{-1}w_1$).

The system clearly has several critical pairs between (anti)monotonicity rules and the remaining rules. Actually, it turns out that every critical pair is not joinable and the system is thus non confluent. Non-joinability is a feature of our system; because our rewriting rules are never applied under a context,from non-joinability it follows that we never compute the same arc twice in different ways.

Computing all normal forms of a term can be done very efficiently (in terms of actual, non asymptotic computational cost of the program): At every step at most two rules can be applied, one produces a stuck term and the other can be implemented as a tail recursive call. It is thus possible to simplify the code of an implementation for a generic term rewriting system.

4 The Advanced Rewriting System

Given a combination $w \in S_n$, the computation of $w\!\downarrow$ and $w\!\uparrow$ by means of the term rewriting system presented in the previous section is very efficient. Nevertheless, the number of combinations to be reduced is exponential in n and the number of reducts for each w is also exponential in n. The limiting factor for the computation of larger and larger approximating graphs is thus the memory required to hold the graph defined by $w\!\downarrow$ and $w\!\uparrow$, which is the initial directed graph G from Definition 3 before the computation of connected components.

To be able to compute larger approximations, we exploit the following result: There exist only 7 distinct equivalence classes of combinations of closure and interior. While this results is well known in the literature and we can obtain it with our technique for very small values of n, we additionally observed that every class can be associated with a regular expression that generates all elements of the class[2]. These seven regular expressions are:

$$\epsilon, i^+, c^+, (ic)^+, (ci)^+, i(ci)^+, c(ic)^+$$

Taking as canonical representatives the shortest expressions in each class, we have that the set of representatives is $\{\epsilon, i, c, ic, ci, ici, cic\}$. Let K be any regular expression that generates the set. When we consider combinations that also contain complement, and remembering that $---x = -x$, we obtain that all combinations can be partitioned into an infinite number of sets of equivalent combinations whose representatives are all generated by the following regular expression E: $(-|--)?(K--?)^*K?$. The set that corresponds to a representative is the set obtained by replacing any occurrence of $-$ with an odd number of occurrences of $-$ and any occurrence of a term generated by K with an element of its equivalence class. For instance $-----icicicic----$ is a member of the set whose representative is $-ic--$. The sets that correspond to different representatives are not distinct according to the $\equiv$ relation. For instance $c-i-$ and $-i-$ are representatives of different sets, but $c-i- \equiv -i-$. Nevertheless, if two elements belong to the same set, than they are equivalent. Thus the $\equiv$ equivalence relation is more fine grained than the equivalence relation that is induced by partitioning with respect to regular expressions.

The idea to speed up our previous algorithm is to avoid to generate the vertices (and relative arcs) that correspond to non-canonical representatives of the equivalence classes discussed above. These vertices will all belong to the connected component that will be collapsed to its canonical representative. For instance, for $n = 7$, our previous algorithm would handle the vertices $\{-, ---, -----, -------\}$ as potentially distinct.

To implement the idea, we change the already presented algorithms in two ways.

[2] This property does not hold any longer when we consider combinations with complement.

`(−−)`	`(cc)`	`(ii)`	`(cici)`	`(icic)`
$--w \rhd w$	$ccw \rhd w$	$iiw \rhd w$	$ciciw \rhd ciw$	$icicw \rhd icw$

`(compatible-1 + i-idempotent)`	`(compatible-2 + c-idempotent)`
$\langle w_1, c{-}iw_2, \leq\rangle \rhd w_1{-}iw_2$	$\langle w_1, i{-}cw_2, \geq\rangle \rhd w_1 i{-}w_2$

Figure 4 Additional rewriting rules.

1. We change the definition of S_n with the following one. The changes apply everywhere in Section 3, and in particular to Definitions 2 and 3.

$$x \in S_n \text{ iff } x \text{ is generated by the regular expression } E \text{ and } |x| \leq n$$

2. We integrate the rewriting system with the rules of Figure 4 after dropping the rule `quasi-idempotent` and the two `idempotent` rules from the previous rewriting system. The reason why we drop these rules is that their left hand side will never match any active configuration due to restricting the definition of S_n.

Considering the rules in Figure 4, we observe that all rules of the first line simplify a combination. Applied repeatedly they put any combination into their K-normal form. The rules of the second line are obtained by applying Knuth-Bendix completion. Note, however, that our rewriting rules come from a non-symmetric relation ($\leq$) and we have to take care of this during the superposition phase of Knuth-Bendix completion. The names of the new rules are a concatenation of the names of the rules superimposed. The new rules are necessary to keep completeness after having changed the definition of S_n. For instance, because $c{-}ii$ does no longer belong to S_4, we are no longer considering the combinations like $(c{-}ii)\Uparrow\ \ni -i$. The new rewriting rule generated by Knuth-Bendix completion takes care of adding $-i$ to $(c{-}i)\Uparrow$ by implicitly performing a step of ii-expansion. Note that, in the original rewriting system, monotonicity of i was only used to perform a step of ii-contraction.

In Figure 4 we only list two rules obtained from the Knuth-Bendix completion because all the others are logically redundant: they allow to derive $w_1 \in w_2\Downarrow$ when there exists a w_3 such that $w_1 \in w_3\Downarrow$ and $w_3 \in w_2\Downarrow$. The redundant rules have been pruned by hand, but it is surely possible to automate the procedure.

The new term rewriting system remains noetherian: all the new rules decrease the length of either the (no longer stuck) combinations or the second component of the active configurations. Of the new rules only those in the first line need to be applied several times in order to obtain the normal form of a term. However, it is easy to show that all critical pairs are joinable. Therefore, by Newman's lemma, the normalisation step implemented by the rules in the first line is confluent, as expected.

▶ **Theorem 7** (Correctness and completeness). *The algorithm based on the advanced rewriting system just described correctly computes the n^{th} approximation of the problem for each n.*

The advanced rewriting system is obtained by rewriting in one step all combinations to their canonical representatives in the equivalence classes identified by the regular expression considered. The same trick can used more aggressively when we build the n^{th}approximation after the $(n-1)^{\text{th}}$. Indeed, we can add to the n^{th}term rewriting system one rewriting rule per combination of length $(n-1)$ that in one step rewrites the combination to its $(n-1)^{\text{th}}$canonical representative.

Since the number of these additional rules is exponential in n, we avoid running the Knuth-Bendix completion, by using the new rules only to normalise terms that are not

$$\frac{}{--=\epsilon}\texttt{(axiom 1)}$$

$$\frac{}{c-=-i}\texttt{(axiom 2)}$$

$$\frac{}{i-=-c}\texttt{(axiom 3)}$$

$$\frac{}{c--=--c}\texttt{(axiom 4)}$$

$$\frac{}{c=-i-}\texttt{(axiom 5)}$$

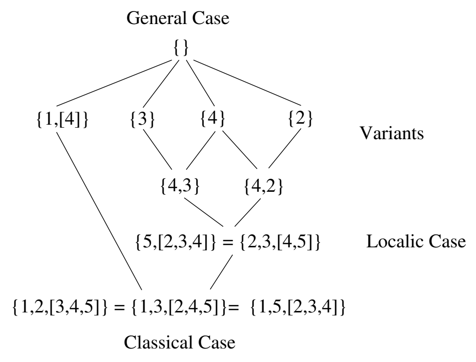


Figure 5 Variations of Kuratowski's problem.

active configurations. The consequence is that we have to normalise exactly the same set of combinations and so we do not save time during the graph generation phase with the rewriting system. The size of the generated graph, however, will be much smaller since it will no longer contain nodes that are not in $(n-1)^{\text{th}}$normal form. The benefit is thus a significant reduction of the computational cost for the computation of the connected components when generating the approximating graph of order n.

The proof of correctness and completeness of the rewriting system obtained with this final improvement is a simple corollary of Theorem 7. The implementation of the improvement is very cheap: the additional rewriting rules generated at the $(n-1)^{\text{th}}$step can only be applied to terms that are stuck according to all other rules. Moreover, they only generate stuck terms. Therefore we can implement this final step as a simple look-up in a trie.

5 More Variations of Kuratowski's Problem

Although the rewriting system presented so far has been developed as a bespoke approach to solve the generalised Kuratowski problem, it turns out that with a parametric implementation our procedure can be applied to a variety of related problems lying between the classical and general problem. These problems are generated by introducing axioms which restrict the general problem, or generalise the classical one. Figure 5 demonstrates variations of Kuratowski's problem, where the axioms on the left hand side gradually refine the generalised problem to the classical problem according to the graph on the right. The nodes are given as sets of included axioms, with the root as the empty set representing the generalised case. Furthermore, axioms derivable from already included ones are given in square brackets.

The variations are motivated by Sambin's work who proposed the generalised problem in the context of intuitionistic point-free topology. Axioms 1–5 are likewise inspired by axioms commonly found in topological problems. For example, axiom 1 postulates the complement operator as idempotent, corresponding to its use in classical logic. Axiom 4, $c--=--c$, is another axiom that is frequently satisfied by concrete basic topologies (see [8]). Adding axiom 5 to the generalised problem, further restricts the saturation operator c. The axiomatisation obtained is the one for locale theory, for which it is already known in the literature [9] that a maximum of 21 combinations exists. Weaker cases than the localic one can be obtained by effectively splitting axiom 5 into axioms 2 and 3, and considering those either separately or in combination with axiom 4.

All the presented problems in the generalised problem domain can be obtained using our approach, by simply adding the corresponding axioms to our advanced term rewriting systems as pairs of reductions over active configurations. The Knuth-Bendix completion must also be applied to combine the new rules with the ones of the advanced term rewriting system.

6 Implementation and Results

Our procedure has been implemented in a combination of bespoke code and existing tools. The rewriting engine has been written in pure OCaml and is fully parametric on the list of reduction rules. The connected-components algorithm exploits the `ocamlgraph` library [2] instantiated with an ad-hoc, optimised hashing function for equivalence classes of combinations. For the transitive reduction of the obtained graph we employ the `tred` tool and for its visualisation we use the `dot` tool.

Implementing the rewrite engine from scratch has allowed us to take care of the peculiarities of the rewriting system, (e.g., by exploiting as much as possible tail recursive calls). This choice was also motivated by the need to generate graph representations that were easy to inspect manually as well as to influence the generation of elements in S in order to explore particular subgraphs, which, to the knowledge of the authors would have been difficult to achieve in any existing system. Furthermore, our implementation allowed us to take some care on memory consumption. Nevertheless, when supplying the rewrite engine with the rules of the advanced rewriting system, the program runs out of memory after about 12 minutes on one of the cores of a server equipped with a 2.4GHz Intel Xeon processor and 48GB of RAM producing an approximating graph of order 16.

That means that it explores all words generated by the regular expression in Section 4 of length at most 16, deriving all equations and inequalities that are provable without using combinations of length ≥ 17. The initial graph generated by the rewriting system contains $1,771,825$ vertices, corresponding to all the combinations of length up to $\equiv_{16}$, and $8,687,605$ arcs, corresponding to steps of the $\leq_n$ relation. The approximating graph obtained after the computation of the strongly connected components contains $44,138$ vertices, corresponding to distinct equivalence classes. The number of arcs in the transitively reduced graph could not be computed as the `tred` tool would not terminate within a 2 hour time-limit already for approximating graphs of order greater than 12. Note that `tred` has been run in separate threads of the computations of our rewrite systems.

The resulting graph is quite chaotic, in that, we were not able to find any simple description of either the set of equivalence classes or the elements of most equivalence classes. Nevertheless, by manual inspection of the generated graph we were able to spot sufficient regularity to solve the problem by showing that the number of equivalence classes is infinite. In fact, all the equivalence classes whose representatives are generated by the following regular expression are distinct: $c?(--c)^*(--)?$. Moreover, each one is less than or equal to every other class generated by a longer representative (e.g. $--c \leq --c--$) and they are all bounded by $-i-$, which is also distinct from them and is the minimum of the lattice.

Figure 6 contains a clipping of the approximating graph of order 12 for the generalised problem visualised with the `dot` tool. The clip contains the approximation of the infinite subgraph with elements of the $c?(--c)^*(--)?$, together with some surrounding nodes. The outgoing arc at the bottom leads to the bottom element of the graph, $-i-$, that is not visible. It is obvious to see that the entire subgraph (i) has only one outgoing arc to the bottom element, (ii) it is less than all elements in the remainder of the graph, and (iii) grows downwards with increasing word length. We briefly sketch the formal argument that leads to the above result using our rewriting formalism.

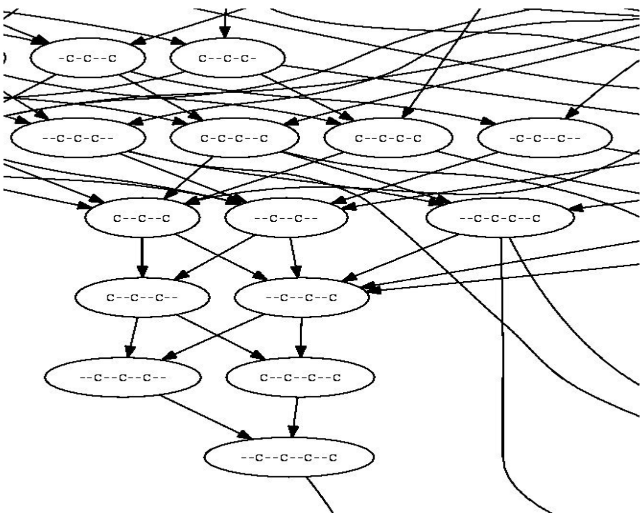


Figure 6 Infinite subgraph for the generalised problem.

First to demonstrate that the equivalence classes generated by the regular expression $r = c?(--c)^*(--)?$ constitute indeed an increasing sequence wrt. $\leq$, we let $\langle w_1, w_2, \geq\rangle$ be any configuration such that $w_2 \neq \epsilon$ and w_1 and w_1w_2 are generated by r.

If $\langle w_1, w_2, \geq\rangle \rhd^* w$ then w is generated by r and is shorter. The argument is by induction over the length $|w_2|$:

1. Suppose w_2 starts with c or with $--$. Then either one of the `saturates` rules is applicable, resulting in a shorter expression.
2. Suppose w_2 starts with c and `monotone` is applicable. Thus w_2 is shorter and we can apply the induction hypothesis.
3. Suppose w_2 starts with $-$ and `antimonotone` is applicable. The new configuration is $\langle w_1-, -w_2', \leq$. The only applicable rule is now `antimonotone` again and we can conclude using the induction hypothesis.

Similarly we can show that $-i-$ is indeed the bottom element: Let $\langle w_1, w_2, \geq\rangle$ be any configuration such that $w_2 \neq \epsilon$ and w_1 and w_1w_2 are generated by r. If $\langle w_1, w_2, \leq\rangle \rhd^* w$ then w is in the same class as $-i-$. Again by induction over $|w_2|$ we can show:

1. Suppose w_2 starts with $c--$ with `compatible-1` we get $w_1-i-w_2 = -i-$.
2. Suppose w_2 starts with c then `monotone` is applicable and we can apply the induction hypothesis.
3. Suppose w_2 starts with $-$ then `antimonotone` is applicable. The new configuration is $\langle w_1-, -w_2', \leq$. The only applicable rule is now again `antimonotone` and we can conclude using the induction hypothesis.

Applying our implementation to other problems in the domain introduced in the previous section, we could quickly verify the results known from the literature of 14 and 21 combinations in the classic and localic case, respectively. For the other problems we obtain a mixed picture of both finite and infinite cases.

Table 1 lists the approximating graphs from order 14 to 16 for the infinite cases in terms of vertices and arcs as well as infinite subgraphs identified. Again no arc count could be computed for the general case due to non-termination of `tred`.

Table 1 Approximating graph for all the variants that do not stabilise.

Axiom set	Order 14		Order 15		Order 16		Infinite
	Classes	Arcs	Classes	Arcs	Classes	Arcs	subgraph
$\emptyset$	10439	?	16869	?	27315	?	$(--c)^*$
$\{2\}$	135	269	142	285	149	299	$(--ci)^*$
$\{3\}$	135	269	142	285	149	299	$(--ic)^*$
$\{4\}$	278	640	283	649	288	660	$(--ici)^*$

So far we have proved formally only the infinite subgraph of the general case to consist of distinct classes. While when adding axiom 2 or 3 or 4 only, the approximating graphs also continue to grow, the infinite subgraph that we spotted in the general case collapses to a finite one as the equation $c-- = --c$ forces all combinations generated by the regular expression $c?(--c)^*(--)?$ into less than four classes. Consequently, the argument we used to show that the general case is infinite does no longer hold. Thus the formal proof for these cases is still outstanding.

For the remaining problem variants the approximating graphs stabilise. The exact figures for the graphs are given in Table 2. Axiom sets $\{1, 2, 3\}$ and $\{2, 3\}$ are the classical and localic case from the literature and we can observe that in our system their approximating graphs stabilise after only few iterations. Similarly, for axiom combinations $\{2, 4\}$ and $\{3, 4\}$, the set of equivalence classes stabilises quickly at 35 and 44 after 8 and 9 iterations, respectively. Finally when adding axiom 1 alone we get a stable approximation after 13 iterations with 126 classes. For these latter three cases we also do not yet have a formal proof because we did not check yet the induced syntactic model.

7 Conclusions and future work

We have presented the study of the generalised version of Kuratowski's classical closure-complement problem from point-set topology. To solve the problem we used a computational procedure that combines a term rewriting system with bespoke graph algorithms. The procedure is capable of showing several million lemmas about relations between combinations of operators, which allowed us to iteratively approximate the solution to the problem. The resulting graph exhibited enough regularity to enable us to show that the solution space of the problem is infinite, thereby successfully closing the problem. A posteriori, the proof that the number of combinations is infinite was quite easy and the infinite set of distinct combinations is generated by a simple regular expression. Nevertheless, the problem has remained open for more than nine years, and the clutter in the rest of the graph made it difficult to spot the infinite subgraph.

Table 2 Approximating graphs for all variants that stabilise.

Axiom set	Classes	Arcs	Stabilising Iteration
$\{1\}$	126	268	13
$\{1, 2, 3\}$	14	16	4
$\{2, 3\}$	21	31	5
$\{2, 4\}$	35	57	8
$\{3, 4\}$	35	57	8

From the mathematical point of view, the result is quite interesting. It shows that the generalisation to a saturation operator partially independent from the reduction operator greatly adds to the expressive power of the system. This is one of the main intuitions at the base of the Basic Picture of Sambin [8], a complete re-formulation of point-wise and point-free topology that is deeply rooted in intuitionistic and predicative logic.

Our graph algorithms, mainly the connected components computation, take care of the transitivity of the less than relation and computes the smallest representatives of the equivalence classes according to the shortlex order. The noetherian term rewriting system where all critical pairs are non joinable takes care of monotonicity and anti-monotonicity of the operators and is used to quickly derive a large number of inferences that do not require transitivity. Non joinability implies that all inferences have different conclusions. Finally, another noetherian and confluent rewriting system allows to aggressively prune the number of inferences by constraining the shape of the terms under consideration. When combined together, we need to apply Knuth-Bendix completion between some rules of the first rewriting system and some of the second one because we implicitly choose the strategy when a term is first simplified using the second set of rules, then reduced using the first, then simplified again. The strategy is implicit since we start from terms already in normal form according to the second set of rules. The completion allows to preserve completeness under the implicit strategy.

Our initial problem that generalises Kuratowski's classical result is not the only possible generalisation. Other typical examples of generalisations are obtained by considering other topological or set theoretical operators like union or intersection. A recent reference for variations on the original problem and applications of the classification to the characterisation of properties of subsets of topological space can be found in [4]. The paper also presents a large bibliography on variants and applications of the problem. It does not cite, however, the generalisation and variants studied in this paper, nor the paper [9] that shows that in locale theory there are exactly 21 different combinations.

We have mapped out a landscape of generalised problems that lie between the finite classical and localic cases, where results were previously known, and the infinite generalised case, which is a new result. We are currently systematising the results about the intermediate cases considered. The general picture obtained, once perfectly clear, will be the subject of a future publication in a mathematical venue.

Already the presented results demonstrate that our approach scales well to all these generalisations. Indeed, the system we implemented is fully parametric on the list of reduction rules of the advanced rewriting system. The computation of the Knuth-Bendix like completion, that at the moment is done manually, could be easily automated. Note, however, that we cannot use any of the many off-the-shelf tools available. The reason is that we need to perform a Knuth-Bendix completion, but we are not dealing with a canonical word problem because the starting relation to be oriented is not symmetric. Rewriting in presence of non symmetric relations has been previously investigated by the third author in [6] to implement semi-equational reasoning and the technique employed here can be effectively understood as an application of this study.

Acknowledgements The authors want to thank Prof. Sambin that stated the generalised Kuratowski's closure-complement problem studied in this paper and personally brought it to the attention of the third author.

References

1 Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, Cambridge, 1998.

2 Sylvain Conchon, Jean-Christophe Filliâtre, and Julien Signoles. Designing a generic graph library using ML functors. In *The Ninth Symposium on Trends in Functional Programming*, volume 8, pages 124–140. Intellect, 2008.

3 Lara Corsi. Combinazioni di operatori di interno, chiusura e loro complemento in LJ. Tesi di laurea in Matematica, Università di Padova, 2006.

4 Barry J. Gardner and Marcel Jackson. The Kuratowski closure-complement theorem. *New Zealand Journal of Mathematics*, 38:9–44, 2008.

5 Kazimierz Kuratowski. Sur l'operation a de l'analysis situs. *Fund. Math.*, 3:182–199, 1922.

6 Claudio Sacerdoti Coen. A semi-reflexive tactic for (sub-)equational reasoning. In *Types for Proofs and Programs*, volume 3839/2006 of *LNCS*, pages 98–114. Springer-Verlag, 2006.

7 Giovanni Sambin. Some points in formal topology. *Theoretical Computer Science*, 305(1-3):347–408, 2003.

8 Giovanni Sambin. *The Basic Picture: a structural basis for constructive topology.* Oxford University Press, 2012.

9 He Wei and Zhang Yasoming. Interior and boundary in a locale. *Advances in Mathematics*, 29(5):439–443, 2000.

Term Rewriting Systems as Topological Dynamical Systems

Søren Bjerg Andersen[1] **and Jakob Grue Simonsen**[2]

1 **Department of Mathematical Sciences, University of Copenhagen**
Universitetsparken 5, DK-2100 Copenhagen Ø, Denmark
`m08sba@math.ku.dk`

2 **Department of Computer Science, University of Copenhagen (DIKU)**
Njalsgade 126, DK-2300 Copenhagen S, Denmark
`simonsen@diku.dk`

Abstract

Topological dynamics is, roughly, the study of phenomena related to iterations of continuous maps from a metric space to itself. We show how the rewrite relation in term rewriting gives rise to dynamical systems in two distinct, natural ways: (A) One in which any deterministic rewriting strategy induces a dynamical system on the set of finite and infinite terms endowed with the usual metric, and (B) one in which the unconstrained rewriting relation induces a dynamical system on sets of sets of terms, specifically the set of compact subsets of the set of finite and infinite terms endowed with the Hausdorff metric.

For both approaches, we give sufficient criteria for the induced systems to be well-defined dynamical systems and for (A) we demonstrate how the classic topological invariant called topological entropy turns out to be much less useful in the setting of term rewriting systems than in symbolic dynamics.

1998 ACM Subject Classification F.1.1 Models of computation, F.4.2 Grammars and other rewrite systems, G.0 General

Keywords and phrases Term rewriting, dynamical systems, topology, symbolic dynamics

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.53

Category Regular Research Paper

1 Topological dynamics and rewriting

A *topological dynamical system* is a pair (X, f) where X is a metric space and $f : X \longrightarrow X$ is a continuous map. The primary object of study is the long-term behaviour of iterations of the map f. Such systems are widely studied in pure and applied mathematics, for example in mathematical physics, in fractal geometry, and in combinatorial number theory [12]. A well-studied subclass of dynamical systems is that of *symbolic dynamical systems* [25] where X is a topologically closed set of infinite strings and f is the *shift map*, either chopping off the first symbol of a right-infinite string or shifting a bi-infinite string one position to the left. Such systems with right-infinite strings can be easily modelled by string rewriting systems with very simple rules ($a \to \epsilon$ for right-infinite strings) and endowed with a rewrite strategy. Equivalently, they can be modelled by term rewriting systems over unary signatures (with rules $a(x) \to x$) endowed with a rewrite strategy. The observation that existing dynamical systems can be modelled by almost trivial rules immediately raises the twin questions of whether more general term rewriting systems on well-behaved sets of terms can be viewed as topological dynamical systems.

We answer this question in the positive by defining two distinct notions of dynamical systems that we believe are quite natural: (A) One in which any (suitably continuous) deterministic rewriting strategy induces a dynamical system on the set of finite and infinite ground terms endowed with the usual metric, and (B) one in which the unconstrained rewriting relation induces a dynamical system on sets of sets of terms, specifically the set of compact subsets of the set of finite and infinite ground terms endowed with the Hausdorff metric. We also provide a starting point for investigating the dynamical properties of these systems by considering continuity, conjugacy and topological entropy.

We employ *infinite, ground* terms for two reasons: (1) that they are needed to generalize the classes of dynamical systems that already exist in the literature, (2) that the set of finite and infinite ground terms is topologically much more well-behaved than the set of finite terms, hence allow stronger results. We stress that the *rewriting relation* itself is ordinary one-step rewriting; thus, we have no infinite reductions, and we do *not* consider infinitary rewriting [18]. However, as topological dynamics considers the long-term behaviour of iterated maps, we do expect that connections with this area will be found in the future.

A clarification from the start: We do not have any practical applications of the present work, nor have we actively sought out such applications. We believe the topological dynamics of term rewriting to be mathematically interesting *per se*, even more so as the systems we consider properly generalize existing classes of dynamical systems.

1.1 Related work

The class of (one-dimensional) symbolic dynamical systems (SDSs) is a special case of the class of systems we define in this paper, and is the main in inspiration for our work; there is a vast literature on SDSs, and we refer the reader to the excellent textbooks and handbook chapters in the area [25, 22, 6]. Kitchens defines three classes of dynamical systems associated to a class of maps on finite trees [23]; the trees are not in general trees of terms and the maps themselves do not correspond to term rewriting. The thematically closest research to ours are the *tree-shifts* of Aubrun and Béal [2, 3], introduced as an intermediate notion between one-sided shifts of dimension one, resp. higher dimensions. In contrast to the systems we consider, tree-shifts pertain to infinite, labelled trees with a fixed arity n (each node in the tree has i children), and the *set* of maps associated to the set of such trees are the n maps σ_i such that σ_i cuts off the root node and returns the ith child of the root. Aubrun and Béal succesfully generalize a number of concepts and results from SDSs to the class of tree-shifts, including the concepts of system of finite type and sofic systems, and prove the highly interesting result that conjugacy of tree-shifts of finite type is decidable. We firmly believe, but have not proved, that these results can be extended to many of the systems we consider in this paper.

2 Preliminaries

2.1 Term rewriting on finite and infinite terms

We refer to standard textbooks [4, 27] for basics on term rewriting; to fix notation, we give the most necessary definitions below.

A *signature* is a set of *function symbols*, each endowed with a non-negative integer *arity*; if f is a function symbol of arity n, we invariably write f/n. If Σ is a signature and V is a set of variables, we denote by $\mathrm{T}(\Sigma, V)$ the set of *terms over* Σ and V; the elements of the set are all variables, all nullary function symbols, and all objects on the form $f(s_1, \ldots, s_n)$

where $f \in \Sigma$ has arity n and $s_1, \ldots, s_n$ are terms over Σ and V. The *root symbol* of a term $s = f(s_1, \ldots, s_n)$ (for $n \geq 0$), denoted root(s), is f. The set of *ground terms over* Σ is $\mathrm{T}(\Sigma, \emptyset)$. A (one-hole) *context* is a term in $\mathrm{T}(\Sigma, V \cup \{\Box\})$ that contains exactly one occurrence of $\Box$, where $\Box \notin \Sigma \cup V$. A *substitution* is a map $\theta : V \longrightarrow \mathrm{T}(\Sigma, V)$ where we usually assume that only a finite number of elements $x \in V$ satisfy $\theta(x) \neq x$. A substitution extends homomorphically to a map $\theta : \mathrm{T}(\Sigma, V) \longrightarrow \mathrm{T}(\Sigma, V)$ in the obvious way. If C is a context and s is a term, we define $C[s] = \theta(C)$ where θ is the substitution defined by $\theta(\Box) = s$ and $\theta(x) = x$ for $x \neq \Box$.

The *set of positions* of a term s, denoted $\mathrm{Pos}\,(s)$ is the set of finite sequences of positive integers defined inductively as follows: If $s \in V$ or s is a nullary function symbol, then $\mathrm{Pos}\,(s) = \{\epsilon\}$ where ϵ is the empty string. If $s = f(s_1, \ldots, s_n)$, then $\mathrm{Pos}\,(s) = \{\epsilon\} \cup \{ip : p \in \mathrm{Pos}\,(s_i) \wedge 1 \leq i \leq n\}$. The *length* of a position is its length as a string. The *subterm* of term s *at position* $p \in \mathrm{Pos}\,(s)$, denoted $s|_p$, is defined inductively by $s|_\epsilon = s$ and otherwise (in which case $s = f(s_1, \ldots, s_n)$ and $p = ip'$ for some $p' \in \mathrm{Pos}\,(s_i)$) by $s|_p = s_i|_{p'}$. For notational convenience, we set $\mathrm{POS} = \mathbb{N}^{<\omega}$ ("the set of all possible positions"), with the convention that $\epsilon \in \mathrm{POS}$. If $p \in \mathrm{POS}$ and j is a non-negative integer, we set $p^0 = \epsilon$ and $p^j = p^{j-1}p$ (i.e., the sequence consisting of j copies of p). If C is a context, we occasionally write C as $C[]_p$ where p is the unique position of $\Box$ in C.

A *term rewriting system* (abbreviated TRS) over Σ and V is a set of pairs, invariably written $l \to r$, such that (i) $l, r \in \mathrm{T}(\Sigma, V)$, (ii) $l \notin V$, and (iii) every variable that occurs in r also occurs in l. The *rewrite relation induced by* R is the binary relation $\to_R \subseteq \mathrm{T}(\Sigma, V) \times \mathrm{T}(\Sigma, V)$ defined by $s \to_R t$ if there exist a rule $l \to r \in R$, a context $C[]_p$ and a substitution θ such that $s = C[\theta(l)]_p$ and $C[\theta(r)]_p = t$ (in which case we say that the pair $(p, l \to r)$ is a *redex* in s). We invariably drop the subscript R, writing $\to$, when there is no risk of confusion. A rule $l \to r$ is called *collapsing* if $r = x \in V$.

The standard *metric* $d : \mathrm{T}(\Sigma, V) \times \mathrm{T}(\Sigma, V) \longrightarrow \mathbb{R}_+ \cup \{0\}$ is defined by $d(s, s) = 0$ and $d(s, t) = 2^{-|p|}$ where p is a position of minimal length such that $\mathrm{root}(s|_p) \neq \mathrm{root}(t|_p)$.

The set of *finite and infinite terms over* Σ and V, denoted $T^\infty(\Sigma, V)$ is the (necessarily unique) metric completion of $\mathrm{T}(\Sigma, V)$. The set of *infinite terms* over Σ and V is $T^\infty(\Sigma, V) \setminus \mathrm{T}(\Sigma, V)$; note that an infinite term may have subterms that are finite. The set of finite and infinite *ground* terms over Σ is $T^\infty(\Sigma, \emptyset)$.

An equivalent definition of $T^\infty(\Sigma, V)$ is obtained by interpreting the term formation rules coinductively instead of inductively, and still other equivalent methods exist in the form of ideal completion and partial functions (see, e.g., [20]) If in doubt: *infinite terms are exactly what you think they are.*

Let R be a TRS over Σ and V; the rewrite relation $\to_R \subseteq T^\infty(\Sigma, V) \times T^\infty(\Sigma, V)$ is defined exactly as in the finite case (note that in all rules $l \to r \in R$, both l and r are still finite terms). We denote by $\mathrm{NF}(R)$ the set of R-normal forms of $T^\infty(\Sigma, \emptyset)$ (i.e. the set of possibly infinite ground terms s such that there exists no t with $s \to t$).

2.2 Topological dynamics

We refer to standard textbooks on topological dynamics such as [12, 28], giving only the most basic definitions below.

A *topological dynamical system* is a pair (X, f) where X is a topological space and $f : X \longrightarrow X$ is a continuous map. It is usually assumed that X is locally compact, metrizable and second countable.

A standard notion of *subsystems* exists for dynamical systems (A, f): A dynamical system (B, f) is a subsystem of (A, f) if B is a topologically closed subset of A that is

f-(forward-)*closed*, that is $f(B) \subseteq B$ [28, Ch. 5].

The following two examples introduce *one-* and *two-sided shifts* that are the primary objects of study in the field of *symbolic dynamics*; for more information, consult textbooks specifically on symbolic dynamics, for example [25, 22].

▶ **Example 2.1** (One-sided shifts). Let $\tilde{\Sigma}$ be a finite string alphabet, and let $F \subseteq \tilde{\Sigma}^*$ be a (finite or infinite) set (called the set of *forbidden words*). Let $\tilde{A}_F$ be the set of all right-infinite strings over $\tilde{\Sigma}$ that do not contain any element of F as a substring. Then by standard results $\tilde{A}_F$ is a topologically closed subset of $\tilde{\Sigma}^\omega$ equipped with the usual Cantor metric on sequences [25]. The *shift map* σ on $\tilde{A}_F$ is defined by $\sigma(b_1 b_2 b_3 \cdots) = b_2 b_3 \cdots$, and it is easily seen that $(\tilde{A}_F, \sigma)$ is a topological dynamical system.

▶ **Example 2.2** (Two-sided shifts). Proceed as in Example 2.1, but define instead $\tilde{B}_F$ as the set of all *bi*-infinite strings over $\tilde{\Sigma}$ that do not contain any element of F as a substring, and let the shift map σ be given by $\sigma(\cdots b_{-2} b_{-1} b_0 b_1 b_2 \cdots) = (c_n)_{n \in \mathbb{Z}}$ where $c_n = b_{n+1}$ for all $n \in \mathbb{Z}$. Then, $(\tilde{B}_F, \sigma)$ is a topological dynamical system.

In both of the above examples we may have $F = \emptyset$ in which case $\tilde{A}_F$ (resp. $\tilde{B}_F$) is the set of all right-infinite (resp. bi-infinite) strings over $\tilde{\Sigma}$.

Finally, recall that a metric space (M, d) is an *ultrametric space* if $d(x, y) \leq \max\{d(x, z), d(z, y)\}$ for arbitrary $x, y, z \in M$. The standard metric d on $T^\infty(\Sigma, \emptyset)$ is an ultrametric.

▶ **Lemma 2.3.** *Let (M, d) be an ultrametric space. If $B(x, \epsilon)$ and $B(z, \delta)$ are open balls with $\delta \leq \epsilon$ and $B(x, \epsilon) \cap B(z, \delta) \neq \emptyset$ then $B(z, \delta) \subseteq B(x, \epsilon)$. In particular, if $\delta = \epsilon$, $B(x, \epsilon) \cap B(z, \delta) \neq \emptyset$ implies $B(x, \epsilon) = B(z, \delta)$.*

Let $A \subseteq M$. Then A has a cover of cardinality k of sets of diameter at most ϵ iff it has a cover of cardinality k of open balls of radius *ϵ.*

Lastly, (M, d_n) is an ultrametric space.

3 Topological dynamics and term rewriting

Many results in topological dynamics are contingent on the underlying topological space being locally compact (and, in symbolic dynamics, compact). It is easy to see that $(\mathrm{T}(\Sigma, V), d)$ is locally compact: For any term s, let m be a positive integer that is strictly greater than the length of the longest position in s; then the open ball with radius 2^{-m} centered on s is exactly the (necessarily compact) singleton set $\{s\}$). However, in many cases we would like $(\mathrm{T}(\Sigma, V), d)$ to be complete–and preferably compact–which it is not in general:

▶ **Lemma 3.1.** *Let Σ contain at least one symbol with arity ≥ 1. If $\mathrm{T}(\Sigma, \emptyset)$ is not empty, then it is not complete (hence is not compact).*

Proof. Let $f \in \Sigma$ have arity $m \geq 1$. If $\mathrm{T}(\Sigma, \emptyset) \neq \emptyset$, there must be $a \in \Sigma$ with arity 0. Define the sequence of terms (s_n) by $s_0 = a$ and $s_{n+1} = f(s_n, \ldots, s_n)$. Then, (s_n) is Cauchy, but has no limit in $\mathrm{T}(\Sigma, \emptyset)$. ◀

If we extend our attention to $T^\infty(\Sigma, V)$, however, the topological properties are much better. The following results hold for the metric space $(T^\infty(\Sigma, V), d)$:

▶ **Lemma 3.2** (See [1]). *Let Σ be a (possibly infinite) signature. Then, $T^\infty(\Sigma, V)$ is complete. In addition, $T^\infty(\Sigma, V)$ is compact iff both Σ and V are finite (in particular, if Σ is finite, then $T^\infty(\Sigma, \emptyset)$ is compact).*

An important convention: In view of Lemma 3.2, we restrict our attention to $T^\infty(\Sigma, \emptyset)$– the set of finite and infinite *ground* terms over a *finite* signature, and hence the compact metric space $(T^\infty(\Sigma, \emptyset), d)$. In addition, unless explicitly otherwise stated, R will always denote a TRS with finitely many rules.

4 Take A: Dynamics on sets of terms with a rewriting strategy

We consider deterministic strategies on the set of finite and infinite terms; intuitively, a strategy picks one among the (possibly infinite number of) redexes in a term to contract:

▶ **Definition 4.1.** Let R be a TRS on $T^\infty(\Sigma, \emptyset)$. A (deterministic) strategy S for R is a (total) map $S : (T^\infty(\Sigma, \emptyset) \setminus \mathrm{NF}(R)) \longrightarrow \mathrm{POS} \times R$ such that $S(t) = (p, l \to r)$ is a redex in t.

If S is a strategy for R, the *map induced by* S, denoted $F_{S,R} : T^\infty(\Sigma, \emptyset) \longrightarrow T^\infty(\Sigma, \emptyset)$ (invariably abbreviated F_S when R is clear from the context), is defined by $F_S(t) = t$ if t is a normal form, and otherwise $F_S(t) = s$ if $t \to s$ by contraction of the redex of rule $l \to r$ at position p where $S(t) = (p, l \to r)$.

There are two potential pitfalls in the above definition: (i) as terms may be infinite, the set $\mathrm{NF}(R)$ of normal forms of R is not in general decidable–even if R is finite–whence computability of F_S is not immediate, (ii) the map F_S need not necessarily be continuous, which is required of a topological dynamical system.

▶ **Definition 4.2.** Let S be a strategy for the TRS R on $T^\infty(\Sigma, \emptyset)$ such that $F_S : T^\infty(\Sigma, \emptyset) \longrightarrow T^\infty(\Sigma, \emptyset)$ is continuous. Then $(T^\infty(\Sigma, \emptyset), F_S)$ is a topological dynamical system and is called the *S-induced dynamical system on* $T^\infty(\Sigma, \emptyset)$.

Thus, the main question of interest is under which circumstances the map F_S is continuous.

Note that if S is *not* continuous as a map $T^\infty(\Sigma, \emptyset) \setminus \mathrm{NF}(R) \longrightarrow \mathrm{POS} \times R$ when POS and R are endowed with the discrete topology and $\mathrm{POS} \times R$ has the product topology, then in general F_S will not be continuous either. For example, consider the signature $\{a/1, b/1\}$, the TRS $\{a(x) \to b(x), b(x) \to a(x)\}$ and the–pathological–strategy S defined by $S(b^k(a(t)) = (1^k, a(x) \to b(x))$ (for any $k \geq 0$ and term t), and $S(b^\omega) = (\epsilon, b(x) \to a(x))$. Then, S is not continuous, as witnessed by the fact that $(b^n(a^\omega))_n$ is sequence converging to b^ω, but $S(b^n(a^\omega)) = (1^n, a(x) \to b(x))$ and $S(b^\omega) = (\epsilon, b(x) \to a(x))$, whereas the sequence $(S(b^n(a^\omega)))$ does not converge to $S(b^\omega)$. Hence, F_S is not continuous.

We have the following useful lemma:

▶ **Lemma 4.3.** *Let R be a (finite!) TRS and let $S : (T^\infty(\Sigma, \emptyset) \setminus \mathrm{NF}(R)) \longrightarrow \mathrm{POS} \times R$ be a strategy for R. Equip R and POS with the discrete topology, and let $\mathrm{POS} \times R$ be equipped with the product topology. If S is continuous, then $F_S : T^\infty(\Sigma, \emptyset) \longrightarrow T^\infty(\Sigma, \emptyset)$ is continuous.*

Proof. We show pointwise continuity at $s \in T^\infty(\Sigma, \emptyset)$. Let k be any non-negative integer and split on cases according to s:

- If s is not a normal form, let $S(s) = (p, l \to r)$. By continuity of S, there is a non-negative integer n_1 such that if $d(s,t) < 2^{-n_1}$, and $S(t) = (p', l' \to r')$, then $p = p'$, $l = l'$ and $r = r'$. Let w be the length of the longest position in l and set $m_1 = \max\{k, n_1, |p| + w\}$. If $d(s,t) < 2^{-m_1}$, we hence have $d(F_S(s), F_S(t)) < 2^{-k}$.
- If s is a normal form, let w be the length of a maximally long position occurring in a left-hand side of a rule in R, and set $m_2 = k + w$. Let $t \in T^\infty(\Sigma, \emptyset)$ such that $d(s,t) < 2^{-m_2}$. If t is not a normal form, consider $S(t) = (q, l \to r)$; if $|q| \leq k$, then there is a redex in s at q, a contradiction. Hence, $d(F_S(s), F_S(t)) < 2^{-k}$. If t is a normal form, then $d(F_S(s), F_S(t)) = d(s,t) < 2^{-m_2} < 2^{-k}$. ◀

Thus, continuity of F_S merely requires us to prove that S is continuous.

▶ **Corollary 4.4.** *Let R be any (finite) TRS, let S be a leftmost-outermost strategy (in case more than one redex is present at a position, assume that S deterministically picks one of them)*[1]*. Then, $(T^\infty(\Sigma, \emptyset), F_S)$ is a topological dynamical system.*

Proof. Let $s \in T^\infty(\Sigma, \emptyset)$ and let k be a non-negative integer. Let w be the length of the longest position occurring in a left-hand side of a rule in R. Set $m = k + w$. If $s, t \in T^\infty(\Sigma, \emptyset)$ are both non-normal forms with $d(s,t) < 2^{-m}$, we reason as follows: For the leftmost-outermost strategy, if S picks a redex in s at position p with $|p| \leq k$, then that redex is present in t as well, and must be leftmost and outermost in t (otherwise another redex present at depth at most k is leftmost-outermost, but then it would also be leftmost-outermost in s, a contradiction). Contracting both redexes yields terms identical up to depth k (as the length of any position across a rewrite step can be *shortened* by at most w). Alternatively, S picks a redex at a position with length strictly greater than k; in this case, S cannot pick a redex at a position of length $\leq k$ in t, as this redex would also be present in s and thus at a position of strictly shorter length than the redex picked for s, contradicting outermostness. In both cases, $d(F_S(s), F_S(t)) < 2^{-k}$, establishing continuity. ◀

Perhaps surprisingly, *innermost* strategies are in general not continuous: Consider the signature $\Sigma = \{a/1, b/1, \rhd/0\}$ and the TRS $R = \{a(x) \to \rhd\}$. Consider terms on the form $a(b^k(a(\rhd)))$ for $k \geq 1$. Then, any innermost strategy will always pick the redex resulting in the step $a(b^k(a\rhd)) \to a(b^k(\rhd))$. But for any term on the form $a(b^k(\rhd))$, any innermost strategy will always pick the redex at the root of the term, resulting in the step $a(b^k(\rhd)) \to \rhd$. If the innermost strategy were continuous, then there would be an integer m such that $d(a(b^k(a(\rhd))), a(b^k(\rhd))) < 2^{-m}$ implies that S chooses the same redex in both terms; but S chooses different redexes in the terms $a(b^m(a(\rhd)))$ and $a(b^m(\rhd))$, a contradiction.

▶ Remark. It turns out that every *computable* strategy S is continuous, but due to space constraints we omit the lengthy definitions needed to define computable strategies on infinite terms. After the definitions have been made, continuity follows immediately from the Kreisel-Lacombe-Shoenfield Theorem ("every computable map on a computably complete computable metric space is continuous") [24]. Note that the leftmost-outermost strategy is computable, but the leftmost-innermost is *not*.

4.1 Examples

Let S be a strategy for TRS R such that the S-induced system $(T^\infty(\Sigma, \emptyset), F_S)$ exists. A *subsystem* of $(T^\infty(\Sigma, \emptyset), F_S)$ is a dynamical system (T', F_S) such that (i) T' is a topologically closed subset of $T^\infty(\Sigma, \emptyset)$, and (ii) $F_S(T') \subseteq T'$

▶ **Example 4.5.** We can easily model the one- and two-sided shifts of Examples 2.1 and 2.2:

Let $\Sigma = \{a_1/1, \ldots, a_k/1\}$. For the one-sided shift, set $R = \{a_1(x) \to x, \ldots, a_k(x) \to x\}$ and let S be the (necessarily unique, as all symbols are unary) outermost strategy for R. For a set of "forbidden words" F, all on the form $a_{i_1}(\cdots(a_{i_n}(x)))$, we may consider the set A_F of infinite, ground terms s such that no element of F occurs in s; formally: there is no context

[1] For the purposes of this paper, a leftmost-outermost strategy is a strategy that picks a redex that is (i) at a position of minimal length, and (ii) is minimal in the lexicographic order on positions among redexes satisfying (i). For further discussion of the difficulties of generalizing leftmost-outermost reduction to infinitary rewriting, see [21].

$C[]$, no element $a_{i_1}(\cdots(a_{i_n}(x))) \in F$, and no $s' \in T^\infty(\Sigma, \emptyset)$ such that $s = C[a_{i_1}(\cdots(a_{i_n}(s')))]$. Then, (A_F, F_S) is a subsystem of $(T^\infty(\Sigma, \emptyset), F_S)$.

Correspondingly, for the two-sided shift, let $u \notin \Sigma$ be a fresh *binary* symbol and define B_F to be the set of all terms on the form $u(s_1, s_2)$ where $s_1, s_2 \in T^\infty(\Sigma, \emptyset)$ such that no string of symbols corresponding to $w \in F$ occurs in s, and no string of symbols corresponding to the *reverse* of $w \in F$ occurs in s_1. Define $R = \{u(x, a_1(y)) \to u(a_1(x), y), \ldots, u(x, a_k(y)) \to u(a_k(x), y\}$ and let S be an outermost strategy for R. Then, (B_F, F_S) is a subsystem of $(T^\infty(\Sigma, \emptyset), F_S)$.

▶ **Example 4.6.** Let $\Sigma = \{: /2, \mathrm{nil}/0, +/1, 0/0, f/1\}$, write : ("cons") as in infix operator as usual, and set $R = \{f(\mathrm{nil}) \to \mathrm{nil}, f(x : y) \to +(x) : f(y)\}$.

Let N be the set of finite and infinite ground terms built solely from $+$ and 0 (i.e., N models the set $\mathbb{N} \cup \{\infty\}$), and let L be the least subset of $T^\infty(\Sigma, \emptyset)$ such that $\mathrm{nil} \in L$, and such that $s \in N$ and $t \in L$ implies $s : t \in L$ (i.e. L models the set of finite and infinite lists of natural numbers). Finally, let $K \subseteq T^\infty(\Sigma, V)$ be the subset of terms such that $s \in K$ implies either $s \in L$, or $s = t_1 : (t_2 : \cdots f(t_n : \cdots))$ where all t_i are elements of N. Then, K is topologically closed, and if F is any outermost strategy, $F_S(s) = s$ for any $s \in K$ and $F_S(s) \in L$ if $s \in K \setminus L$. Finally, as there is at most one redex in any element of K, F_S is continuous as S is an outermost strategy.

The map F_S contracts the unique redex, if any, in a term $s \in K$. By inspection, we see that iteration of the map corresponds to "adding one" to each natural number in successive elements of the list s. If $s = f(s')$ where $s' \in L$, the sequence $F^n(s)$ converges to the (finite or infinite) term $t \in K$ with one added to every natural number in s.

5 Take B: Dynamics on sets of sets of terms

We now consider the action of maps F_R induced by R where the objects being "moved" by F_R are *set* of terms. Hence, the natural dynamical system is a map on a set of sets of terms.

▶ **Definition 5.1.** Let (X, d) be a metric space and let $\mathcal{H}(X)$ be the set of non-empty compact subsets of X. The *Hausdorff metric* on $\mathcal{H}(X)$ is then the metric d_H defined by

$$d_H(A, B) = \max\left\{\sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b)\right\}$$

A proof that d_H is indeed a metric on $\mathcal{H}(X)$ can be found for example in [5]. It is not hard to see that if (X, d) is an ultrametric space, then so is $(\mathcal{H}(X), d_H)$, and that completeness of (X, d) implies completeness of $(\mathcal{H}(X), d_H)$. For the standard metric on terms, there is a simple, neat characterization of the Hausdorff metric:

▶ **Theorem 5.2** (See e.g. Ch. 1 of [13]). *Let x be a variable and define, for any non-negative integer n and any $s \in T^\infty(\Sigma, \emptyset)$, the term $\alpha_n(s)$ as the term s where we have replaced any subterm of s at positions p with $|p| > n$ by x. For $A \in \mathcal{H}(T^\infty(\Sigma, \emptyset))$, define $\alpha_n(A) = \{\alpha_n(s) : s \in A\}$.*

The Hausdorff metric on $T^\infty(\Sigma, \emptyset)$ is then given by

$$d_H(A, B) = 2^{-\inf\{n : \alpha_n(A) \neq \alpha_n(B)\}}$$

▶ **Definition 5.3.** Let R be a TRS. We then define the map $F_R : \mathcal{H}(T^\infty(\Sigma, \emptyset)) \longrightarrow \mathcal{H}(T^\infty(\Sigma, \emptyset))$ by

$$F_R(A) = \{t \in T^\infty(\Sigma, \emptyset) : s \in A \wedge s \to_R t\} \cup \{t \in A : t \text{ is a normal form}\}$$

If the map F_R is continuous, then $(\mathcal{H}(T^\infty(\Sigma, \emptyset)), F_R)$ is a dynamical system, which we denote the *R-induced system.*

Thus, $F_R(A)$ consists of (i) all reducts of elements of A, and (ii) all normal forms in A. The reason for including the normal forms in A is to ensure that F_R is well-defined (as, otherwise, any set consisting solely of normal forms would be mapped into the empty set, which is not an element of $\mathcal{H}(X)$).

Fortunately, F_R is continuous for all ordinary TRSs R:

▶ **Theorem 5.4.** *If R has finitely many rules, then F_R is continuous.*

Proof. Observe that $d_H(A, B) < \epsilon$ iff

$$(\forall s \in A \exists t \in B.d(s,t) < \epsilon) \wedge (\forall t \in B \exists s \in A.d(s,t) < \epsilon)$$

Let m be the length of a position of maximal length occurring in the left-hand side of a rule of R, and let k be an arbitrary non-negative integer. To show that F_R is continuous, it suffices to show that:

1. $\big(\forall s \in A \exists t \in B.d(s,t) < 2^{-(2k+m)}\big) \Rightarrow \big(\forall s' \in F_R(A) \exists t' \in F_R(B).d(s',t') < 2^{-k}\big)$, and
2. $\big(\forall t \in B \exists s \in A.d(t,s) < 2^{-(2k+m)}\big) \Rightarrow \big(\forall t' \in F_R(B) \exists s' \in F_R(A).d(t',s') < 2^{-k}\big)$.

Note that the two conditions above are completely symmetrical in A and B; we hence prove only the first.

Assume that $\forall s \in A \exists t \in B.d(x,y) < 2^{-(2k+m)}$ and let $s' \in F_R(A)$ be arbitrary. Then there exists $s \in A$ such that either (i) $s = s'$ and s is a normal form, or (ii) $s \to_R s'$.

In case (i), consider $t \in B$: If t is a normal form, then $t \in F_R(B)$ and we may set $t' = t$, and we have $d(s',t') = d(s,t) < 2^{-(2k+m)} \leq 2^{-k}$, as desired. If t is not a normal form, then $t \to t'$ for some term t' by a rewrite step at some position p; note that $|p| \geq k$ as s is a normal form and s and t are identical up to depth $2k+m$. Hence, $d(s',t') = d(s,t') < 2^{-k}$, as desired.

In case (ii), let p be the position of the redex contracted in the step $s \to s'$. Split on cases as follows:

- If $|p| < k$, note that $d(s,t) < 2^{-(2k+m)}$, whence there is a redex of the same rule at p in t; let t' be the term such that $t \to t'$ by contraction of this redex, and observe that $t' \in F_R(B)$. Observe that the rule employed in this redex can *shorten* positions by at most m; hence, if the redex is contracted at depth $|p| < k$, the symbols occurring at depth $\leq k$ in t' are either created by the right-hand side of the rule, or are descendants of positions occurring at depth at most $2k+m$. As s and t were identical up to depth $2k+m$, t' and s' are thus identical up to depth k, whence $d(s',t') < 2^{-k}$, as desired.
- If $|p| \geq k$, then as d is an ultrametric, we have $d(s',t') \leq \max\{d(s',s), d(s,t')\} \leq \max\{d(s',s), \max\{d(s,t), d(t,t')\}\} < 2^{-k}$. ◀

The assumption that R has a finite number of rules cannot be omitted from the statement of Theorem 5.4 as there are systems R with infinitely many rules for which F_R is not continuous, cf. the following example.

▶ **Example 5.5.** Consider the signature $\Sigma = \{a/1, b/1\}$ and $R = \{a^n(b(x)) \to b(x) : n \in \mathbb{N}_0\}$. Define $A = \{a^\omega\}$ and, for each nonnegative integer k, $B_k = \{a^k(b^\omega)\}$. All sets A and B_k are singletons, hence non-empty and compact, whence $A, B_k \in \mathcal{H}(T^\infty(\Sigma, \emptyset))$ for all k. For arbitrary k, we have $d_H(A, B_K) = 2^{-k}$, and $(B_k)_k$ thus converges to A. If F_R

were continuous, the sequence $(F_R(B_k))_k$ would converge to $F(A)$. But $F_R(A) = A$ and $F_R(B_k) = \{b^\omega, a(b^\omega), \ldots, a^k(b^\omega)\}$ for each k, whence $d_H(F_R(A), F_R(B_k)) = 1$ for all k, disproving continuity.

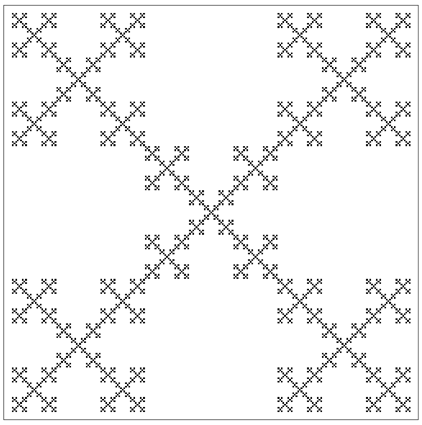
5.1 Examples

As the dynamical system $(\mathcal{H}(T^\infty(\Sigma, \emptyset)), F_R)$ tracks the evolution of sets of terms while encompassing all possible rewrite steps in the terms, it is natural to consider the dynamical system applied to processes usually described by non-deterministic evolution where the set of *all* possible trajectories is the point of interest. One such area is fractals; we show a single example below.

▶ **Example 5.6.** Let $\Sigma = \{0/0, 1/1, f/9\}$, and let R consist of the single rule $1 \to f(1, 0, 1, 0, 1, 0, 1, 0, 1)$. Consider the set $T' = T^\infty(\Sigma, \emptyset) \setminus \{0\}$. T' is a closed subset of the compact set $T^\infty(\Sigma, \emptyset)$, hence compact. Then $(\mathcal{H}(T'), F_R)$ is a topological dynamical system (indeed, a subsystem of $(\mathcal{H}(T^\infty(\Sigma, \emptyset)), F_R)$).

It is easy to see that $F_R|_{\mathcal{H}(T')} : \mathcal{H}(T') \longrightarrow \mathcal{H}(T')$ satisfies $d_H(F_R(A), F_R(B)) = d_H(A, B)/2$ for arbitrary non-empty compact subsets of T'. Hence, by the Banach Fixed Point Theorem, F_R has a unique fixed point given by $\lim_{n\to\infty} F_R^n(A)$ where A is any element of $\mathcal{H}(T')$. In particular, we may choose $A = \{1\}$ and hence see that the fixed point is the term $t_f = \lim_{n\to\infty} t_n$ where $t_0 = 1$ and $t_n = f(t_{n-1}, 0, t_{n-1}, 0, t_{n-1}, 0, t_{n-1}, 0, t_{n-1})$.

It is convenient to visualize t_f by letting the root represent a square with edges of unit length. If this square is partitioned into 9 smaller squares with edge length 1/3 in the obvious manner, the 9 subterms of the root are then represented by one of the smaller squares, and so forth. This generates the box fractal as shown to the right. (More precisely, the box fractal can be drawn as follows: Define $f : \{1, \ldots, 9\} \to [0, 1] \times [0, 1]$ by $f(i) = \frac{1}{3}(\lfloor \frac{i-1}{3} \rfloor, (i - 1) \mod 3)$. For each $p = \epsilon p_1 \cdots p_n \in Pos(t_n)$ with $t_n|_p = 1$ compute $c_p = \sum_{i=1}^{n} f(p_i)/(3^i)$ and draw a square with lower left corner c_p and side length $1/(3^{n+1})$.)



It is tempting to try to derive a general result from Example 5.6. However, the use of the Banach Fixed Point Theorem (which requires the map F_R to be a contraction, i.e., $d(F_R(A), F_R(B)) < d(A, B)$ for all A, B with $A \neq B$), is problematic from the vantage point of term rewriting: If R has at least two distinct normal forms $t \neq t'$, then F_R is not a contraction, due to $F_R(\{t\}) = \{t\}$ and $F_R(\{t'\}) = \{t'\}$, whence $d(F_R(\{t\}), F_R(\{t'\})) = d(\{t\}, \{t'\})$. The obvious “fix” for this problem is to consider F_R as a map on the set of compact subsets on $T^\infty(\Sigma, \emptyset) \setminus \mathrm{NF}(R)$; but this attempt fails as removing the set of normal forms in general destroys compactness[2].

[2] More precisely: If there is a normal form $t \in T^\infty(\Sigma, \emptyset)$ such that (i) t is an infinite term, (ii) there is a sequence (t_n) with $t_n \in T^\infty(\Sigma, \emptyset) \setminus \mathrm{NF}(R)$ for all n and $t = \lim_n t_i$, then $T^\infty(\Sigma, \emptyset) \setminus \mathrm{NF}(R)$ is not complete, hence not compact.

Another example of use is to consider a construction close to the tree-shifts of Aubrun and Béal under the action of all shift maps:

▶ **Example 5.7.** Let Σ be any finite, non-empty signature where each symbol has the same, positive arity (we choose $\Sigma = \{f/2, g/2\}$ in this example). Consider the system $R = \{f(x_1, \ldots, x_n) \to x_i : f \in \Sigma, 1 \leq i \leq n\}$. Then, $F_R : \mathcal{H}(T^\infty(\Sigma, \emptyset)) \longrightarrow \mathcal{H}(T^\infty(\Sigma, \emptyset))$ is continuous by Theorem 5.4. If $A = T^\infty(\Sigma, \emptyset)$, we have $F_R(A) = A$; if B is the subset of $T^\infty(\Sigma, \emptyset)$ consisting of all those infinite ground terms that do not contain the pattern $f(g(\cdot,\cdot), g(\cdot,\cdot))$, then B is compact and $F_B(B) = B$.

6 Conjugacy and topological entropy

Dynamical systems are usually identified up to *topological conjugacy* (roughly: The dynamical properties of two conjugate systems are the same). As topological conjugacy is, in general, undecidable, even for finitely presented dynamical systems, a number of *topological invariants* (quantities known to be identical for conjugate systems) are studied with the purpose of proving that two distinct dynamical systems are *not* conjugate; the most well-known of these being the so-called *topological entropy.* We briefly define these concepts for S-induced systems and show that the topological entropy is unlikely to be a useful topological invariant in this case.

▶ **Definition 6.1.** Two dynamical systems (X, f) and (Y, g) are said to be *topologically conjugate* (or simply *conjugate*) if there is a homeomorphism $h : X \longrightarrow Y$ such that $h \circ f = g \circ h$.

▶ **Example 6.2.** For S-induced systems, systems obtained by simple renamings are conjugate; e.g. if $\Sigma_1 = \{a/1, 0/0\}$, $\Sigma_2 = \{b/1, 1/0\}$, $R_1 = \{0 \to a(0)\}$ and $R_2 = \{1 \to b(1)\}$, then there is only a single strategy S_1 for R_1 and a single strategy S_2 for R_2, and $(T^\infty(\Sigma_1, \emptyset), F_{S_1})$ and $(T^\infty(\Sigma_2, \emptyset), F_{S_2})$ are conjugate.

Likewise, let $\Sigma = \{f/2\}$, $R_1 = \{f(x, y) \to x\}$ and $R_2 = \{f(x, y) \to y\}$, and define, for any $t \in T^\infty(\Sigma, \emptyset)$, the term $h(t) \in T^\infty(\Sigma, \emptyset)$ by swapping left- and right-subterms of each occurrence of f top-down. Then $H : \mathcal{H}(T^\infty(\Sigma, \emptyset)) \longrightarrow \mathcal{H}(T^\infty(\Sigma, \emptyset))$ defined by $H(A) = \{h(t) : t \in A\}$ is a homeomorphism and F_{R_1} and F_{F_2} are conjugate by H.

▶ **Example 6.3.** The one- and two-sided shifts defined in Examples 2.1 and 2.2 are conjugate to their term rewriting counterparts of Example 4.5—for the two-sided shift, define $h(\cdots b_{-2}b_{-1}b_0b_1b_2\cdots) = u(b_{-1}(b_{-2}(\cdots)), b_0(b_1(b_2(\cdots))))$.

▶ Remark. Unsurprisingly, the question of whether two, suitably presented, subsystems are conjugate, is undecidable:

Let Σ be a signature with at least two distinct symbols both of which have arity ≥ 1. The following question is undecidable:

- *Given:* (i) Two finite TRSs R and R' (it may wlog. be assumed that $R = R'$), (ii) two strategies S and S' (it may wlog. be assumed that $S = S'$) for resp. R and R', and (iii) two recursively enumerable, topologically closed subsets $A, A' \subseteq T^\infty(\Sigma, \emptyset)$ that are resp. F_S and $F_{S'}$-invariant.
- *To decide:* Is (A, F_S) conjugate to $(A', F_{S'})$?

Proof. Let $\Sigma = \{0/1, \rhd/0\}$, set $R = R' = \{0(x) \to x\}$ and let $S = S'$ be the outermost strategy. Let M be an inputless Turing machine and define the set

$$A_M = \{0^\omega\} \cup \{0^k \rhd : M \text{ has not halted in the first } k \geq 0 \text{ steps of its execution}\}$$

Note that A_M is finite iff M halts after some number of steps and hence that A_M is closed in this case. If M does not halt, then A_M is infinite and $A_M = \{\rhd, 0\rhd, 00\rhd, 000\rhd, \ldots\} \cup \{0^\omega\}$, whence A_M is also closed in this case. Let S be the (unique) outermost strategy for R. Then $F_S(0^\omega) = 0^\omega$ and, for $k \geq 1$, $F_S(0^k\rhd) = 0^{k-1}\rhd$, whence $F_S(A_M) \subseteq A_M$ in all cases.

Let N be any non-halting Turing machine. Then $A_N = \{0^k\rhd : k \geq 0\} \cup \{0^\omega\}$. Note that as A_M is finite for any halting M and a conjugacy $h : A_M \longrightarrow A_N$ must be a homeomorphism (in particular must be a bijection), then (A_M, F_S) and (A_N, F_S) are conjugate iff M does *not* halt. Conversely, if M does not halt, then (A_M, F_S) and (A_N, F_S) are clearly conjugate (as $A_M = A_N$ and the two systems have the same rule set and strategies).

If it were decidable for all (A_M, R) whether (A_M, F_S) and (A_N, F_S) are conjugate, then we could decide whether M halts as the two systems are conjugate iff M does not halt, and we obtain a contradiction. ◀

We do not know whether it is decidable in general whether two systems on the full sets of terms over (possibly distinct) signatures are conjugate, whether conjugacy is decidable for subsystems (A, F_S) with decidable A. We very strongly suspect that both of these questions are undecidable as well.

A similar result holds for subsystems of $(\mathcal{H}(T^\infty(\Sigma, \emptyset)), F_R)$; the proof uses a construction very similar to the one above and is thus omitted:

The following question is undecidable:

- *Given:* (i) Two finite TRSs R and R' (it may wlog. be assumed that $R = R'$) and (ii) two recursively enumerable, topologically closed subsets $\mathcal{A}, \mathcal{A}' \subseteq \mathcal{H}(T^\infty(\Sigma, \emptyset))$ (both consisting of finite subsets) that are resp. closed under F_R and $F_{R'}$.
- *To decide:* Is $(\mathcal{A}, F_R)$ conjugate to $(\mathcal{A}', F_{R'})$?

6.1 Topological entropy

A topological *invariant* is a quantity that is equal for conjugate systems. Such invariants are used to prove that certain systems are *not* conjugate. The topological entropy, defined below, is a common such invariant (see, e.g., [12, Cor. 2.5.4] for a proof its invariance).

▶ **Definition 6.4.** Let (X, d) be a compact metric space and $f : X \longrightarrow X$ be continuous. For positive integer n, define $d_n(x, y) = \max_{0 \leq j \leq n-1} d(f^j(x), f^j(y))$. For $\epsilon > 0$, let $\text{cov}(n, \epsilon, f)$ be the *minimum* number of sets of d_n-diameter at most ϵ whose union contains X (the d_n-diameter of a set A is the quantity $\sup_{x,y \in A} d_n(x, y)$).

The *topological entropy* of f, denoted $h(f)$, is then:

$$h(f) = \lim_{\epsilon \to 0+} \limsup_{n \to \infty} \frac{1}{n} \log \text{cov}(n, \epsilon, f)$$

The definition of topological entropy can be intuitively understood as follows: Imagine a computer screen with a picture of A as a two-dimensional set; the map f then defines how the points in A move in one time step, and $d_n(x, y)$ measures the maximum distance of the trajectories of the points $x, y \in A$ after the first $n - 1$ time steps; ϵ can be viewed as the "resolution" of our computer screen (small ϵ gives high resolution). The topological entropy is then, roughly, the (exponential) rate of evolution of the number of distinct trajectories we can discern as the number of time steps becomes very large and our resolution becomes very high.

▶ **Example 6.5.** Let $\Sigma_1 = \{f_1/2, \ldots, f_N/2\}$ where $N \geq 1$ and $R_1 = \{f_1(x_1, x_2) \to f_2(x_1, x_2), \ldots, f_N(x_1, x_2) \to f_1(x_1, x_2)\}$, and let S_1 be the (necessarily unique) outermost strategy for R_1. Then, F_{S_1} is an isometry, whence $d_n(s,t) = d(s,t)$ for all non-negative integers n and all $s, t \in T^\infty(\Sigma, \emptyset)$. But for each $\epsilon > 0$, the minimum number of sets of d-diameter sets needed to cover X is then a number, k, independent of n, whence $h(F_{S_1}) = \lim_{\epsilon \to 0+} \limsup_{n \to \infty} 1/n \log k = 0$.

Let $\Sigma_2 = \{f_1/1, \ldots, f_N/1\}$ where $N \geq 1$ and $R_2 = \{f_1(x) \to x, \ldots, f_N(x) \to x\}$, and S_2 be the (again, unique) outermost strategy for R_2. Then, F_{S_2} is conjugate to the full one-sided shift on N symbols, hence has identical entropy. By standard results (see, e.g., [17, p. 120]), the topological entropy of this system is $\log N$; hence $h(F_{S_2}) = \log N$.

Let $\Sigma_3 = \{f_1/n_1, \ldots, f_N/N_n\}$, let R_3 be an orthogonal TRS containing at least one rule, and such that each rule is on the form $f_i(x_1, \ldots, x_{n_i}) \to r$ where every variable in r occurs at depth at least 2. Let S_3 be the leftmost-outermost strategy for R_3. Then, for all $s, t \in T^\infty(\Sigma, \emptyset)$, we have $d(F_{S_3}(s), F_{S_3}(t)) \leq d(s,t)$, and by arguing as for R_1 above, we obtain, *mutatis mutandis*, that $h(F_{S_3}) = 0$.

A moment's thought reveals that when f is a weak contraction (i.e., $d(f(x), f(y)) \leq d(x,y)$ for all $x, y \in X$), then $h(f) = 0$, and as Example 6.5 shows, this situation may occur if the depth of any occurrence of a variable is not decreased in any rule. For depth-*decreasing* systems such as R_2 in Example 6.5, the situation at first glance seems more promising as we obtain positive, finite entropy. However, we will momentarily show that this is an artifact of all function symbols in the signature being unary. In general, systems containing at least two symbols of arity at least 2, and just a single collapsing rule with at least two variables in the left-hand side will have infinite entropy (for outermost strategies).

Before we proceed, we need an ancillary notion: If A is a closed subset of (X, d) such that $F(A) \subseteq A$, the restriction $f|_A$ of f to A induces a topological dynamical system, and we may hence consider the topological entropy $h(f|_A)$. The following result is standard (see, e.g., [12, Prop. 2.5.5(3)]):

▶ **Lemma 6.6.** *Let (X, d) be compact and let $f : X \longrightarrow X$ be continuous. If $A \subseteq X$ is closed and satisfies $F(A) \subseteq A$, then $h(f|_A) \leq h(f)$.*

We then have:

▶ **Theorem 6.7.** *Let Σ be a signature containing at least two elements $f, g \in \Sigma$ each of which has arity at least 2. Let $l \to x$ be a rule where l is a linear term (i.e. each variable occurs at most once) containing at least two distinct variables x and y.*

Let $R \supseteq \{l \to x\}$ be a TRS where no rule in $R \setminus \{l \to x\}$ overlaps $l \to x$ at the root, and let S be any outermost strategy for R. Then, $h(F_S) = \infty$.

Proof. Let p_x and p_y be the unique positions of x and y in l. Define the map $g_l : \mathrm{T}(\Sigma, V) \longrightarrow \mathrm{T}(\Sigma, V)$ by $g_l(s) = l[s]_{p_x}$. As l is not a variable, the sequence $(g_l^n)_n$ is convergent in $T^\infty(\Sigma, V)$ for every s and must converge to an infinite term. Thus, define $t_l = \lim_{n \to \infty} g_l^n(l)$. Note that t_l is an infinite term that is *not* ground, as l contains at least one variable distinct from x.

Let $A \subseteq T^\infty(\Sigma, \emptyset)$ be the set of infinite ground terms such that $s \in A$ iff (i) for each position p of t_l such that $t_l|_p \notin V$, we have $t_l|_p = s|_p$, and (ii) if q is a position in s not covered by (i), then the root symbol of $s|_q$ is either f or g. (Intuition: s is obtained by starting with t_l and then filling in infinite ground terms over the signature $\{f/2, g/2\}$ at all occurrences of variables in t_l.) Let (s_n) be a convergent sequence of elements of A. Clearly, the limit of the sequence must satisfy (i) and (ii) above, whence A is a closed set. For $s \in A$, we permit ourselves to talk about the "stacked copies of l" when referring to the copies of l in the "spine" t_l that is present s.

Observe that, as l was a linear term, each element of A contains a redex of rule $l \to x$ at the root. As no rule in R distinct from $l \to x$ overlaps $l \to x$ at the root, any outermost strategy S must, when applied to $s \in A$, always select the redex of rule $l \to x$ at the root. By construction of A, contraction of this redex yields an element of A (specifically, we have $F_S(s) = s|_{p_x}$), whence $F_S(A) \subseteq A$.

By Lemma 6.6 it thus suffices to prove that $h(F_S|_A) = \infty$.

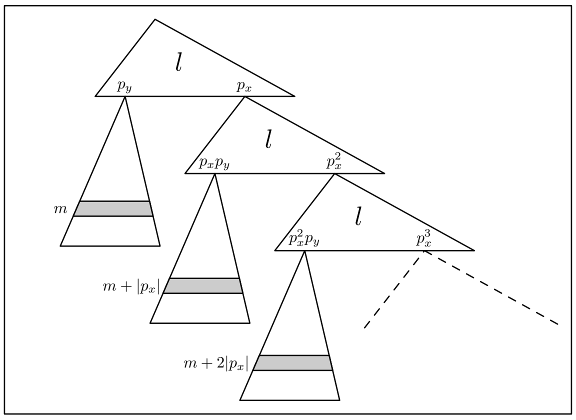
Let p be any position, $j \geq 0$, and consider p_x^j; by the above observations, we have $F_S^j(s) = s|_{p_x^j}$, and hence for any position p in $F_S^j(s)$, we have $F_S^j(s)|_p = s|_{p_x^j p}$. Thus, for arbitrary $s, t \in A$:

$$d_n(s,t) = \max_{0 \leq j \leq n-1} \max\{2^{-|p|} : \mathrm{root}(s|_{p_x^j p}) \neq \mathrm{root}(t|_{p_x^j p})\}\,.$$

(The second max in the above is due to the fact that the position p of *minimal* length satisfying the conditions defines the metric; for such a p the quantity $2^{-|p|}$ is *maximized.*)

By Lemma 2.3, d_n is an ultrametric, and we may thus compute the topological entropy by counting balls of radius ϵ instead of sets of diameter ϵ. Also by Lemma 2.3, two open d_n-balls $B_n(s, 2^{-m})$ and $B_n(t, 2^{-m})$ are either disjoint or equal, and we can thus compute $\mathrm{cov}(n, 2^{-m}, F_S)$ by counting the number of distinct balls of radius 2^{-m}. We proceed by giving a lower bound on the number of such balls.

Let B be the set of infinite ground terms built solely of symbols f and g such that $t \in B$ iff $\mathrm{root}(t|_p) = f$ for $|p| \neq m - |p_y|$. (i.e., t "consists of fs at all depths except $m - |p_y|$ where t may use both f and g"). Observe that $|B| = 2^{2^{m-|p_y|}}$ (there are exactly $2^{m-|p_y|}$ positions of length $m - |p_y|$, and each of these can hold either f or g). Let $s \in A$ be arbitrary and set $C_{m,1} = \{s[t]_{p_y} : t \in B\}$, resp. $C_{m,n} = \{s_n[t]_{p_x^{n-1} p_y} : t \in B \wedge s_n \in C_{m,n-1}\}$. (Intuition: In the first n stacked copies of l, we replace the subterm at p_y by an element of B. See the drawing to the right.) Observe that $|C_{m,1}| = 2^{2^{m-|p_y|}}$ $|C_{m,n}| = 2^{2^m-|p_y|} \cdot |C_{m,n-1}|$, whence $|C_{m,n}| = 2^{n(2^m-|p_y|)}$. Let $s, s' \in C_{m,n}$ with $t \neq t'$. Observe that $\mathrm{root}(s|_p) = \mathrm{root}(s'|_p)$ for all positions p *not* on the form $p = p_x^j p_y q$ where $|q| = m - |p_y|$.



Thus, there must be a position $p = p_x^j p_y q$ with $|q| = m - |p_y|$ such that $\mathrm{root}(s|_p) \neq \mathrm{root}(s'|_p)$. Let p have minimal length among such positions. Observe that $F_S(s) = s|_{p_x}$ and $F_S(s') = s'|_{p_x}$; thus, $d_n(s, s') = 2^{-|p_y q|} = 2^{-m}$.

All open d_n-balls $B_n(s, 2^{-m})$ with $s \in C_{m,n}$ are disjoint by Lemma 2.3. Note that all of these balls must occur in *any* minimal cover of A by balls of radius 2^{-m} (for, if some ball $B_n(t, 2^{-m})$ of radius 2^{-m} contains $s \in C_{m,n}$, then $B_n(t, 2^{-m}) \cap B_n(s, 2^{-m}) \neq \emptyset$, whence by Lemma 2.3 we have $B_n(t, 2^{-m}) = B_n(s, 2^{-m})$.

Hence, $\mathrm{cov}(n, 2^{-m}, F_S) \geq |C_{m,n}| = 2^{n2^{m-|p_y|}}$. Thus:

$$\begin{aligned} h(F_S|_A) = \lim_{\epsilon\to 0+} \limsup_{n\to\infty} \frac{1}{n} \log \mathrm{cov}(n,\epsilon,F_S|_A) &\geq \lim_{m\to\infty} \limsup_{n\to\infty} \frac{1}{n} \log 2^{n2^{m-|p_y|}} \\ = \lim_{m\to\infty} \limsup_{n\to\infty} \frac{n2^{m-|p_y|}}{n} \qquad &\geq \lim_{m\to\infty} 2^{m-|p_y|} = \infty\,, \end{aligned}$$

and the result follows. ◀

Thus, if for example $\Sigma = \{f/2, g/3, h/0\}$ and $R = \{f(x,y) \to y, g(x,y,z) \to g(f(x,y),h,h)\}$ and S is the outermost-left strategy, then $h(F_S) = \infty$.

Theorem 6.7 and Example 6.5 together show that the problems with collapsing rules already known in *infinitary* rewriting rear their heads in our setting of finite reductions as well; this is not entirely surprising as the topological entropy concerns the limiting behaviour of the one-step rewrite relation. The problems with entropy are reminiscent of the result from infinitary rewriting that an orthogonal system is confluent iff it is *almost non-collapsing*, that is, has at most one collapsing rule, and such that the unique collapsing rule has exactly one variable in the left-hand side [19].

7 Conclusion and future work

We have laid the groundwork for the study of topological dynamical systems induced by term rewriting systems; while we have only scratched the surface, we have shown that such systems properly generalize well-known classes of systems such as symbolic dynamical systems, and are equipped with very interesting dynamical properties of their own.

A plethora of open questions remain. We mention a few and give suggestions for future work:

- Is conjugacy undecidable for rewriting systems over the *full* set of terms (i.e., without passing to a closed proper subset of $T^\infty(\Sigma, \emptyset)$? Conjecture: yes.
- There is a rich interaction between topological dynamical systems and measure theory; in the same vein, it is highly conceivable that there are links between our work and probabilistic rewriting [9, 10, 11].
- There are several properties of topological dynamical systems that are typically studied for each class of such systems, for example topological transitivity, mixing and expansiveness [12]; it would be interesting to obtain characterizations of the term rewriting systems whose induced topological dynamical systems have these properties.
- The computability of the dynamics of general dynamical systems is well-studied (see, e.g., [14, 15, 16]), as is complexity issues related to certain aspects of dynamical systems (a few examples: [8, 7, 26]). It would be of interest to perform similar investigations of computability and complexity for the systems defined in this paper.
- There is a need for better topological invariants than the topological entropy; in addition, the entropy of F_R should be investigated.
- What are the connections to the field of infinitary rewriting?

Acknowledgements The authors thank Jeroen Ketema and the anonymous referees for helpful comments.

References

1 A. Arnold and M. Nivat. The metric space of infinite trees. Algebraic and topological properties. *Fundamenta Informaticae*, 3(4):445–476, 1980.

2 N. Aubrun and M.-P. Béal. Decidability of conjugacy of tree-shifts of finite type. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, and W. Thomas, editors, *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 2009.

3 N. Aubrun and M.-P. Béal. Sofic and almost of finite type tree-shifts. In F. M. Ablayev and E. W. Mayr, editors, *CSR*, volume 6072 of *Lecture Notes in Computer Science*, pages 12–24. Springer, 2010.

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

5 M. Barnsley. *Fractals Everywhere*. Morgan Kaufmann, 1993.

6 M.-P. Béal and D. Perrin. Symbolic dynamics and finite automata. In G. Rosenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 2, chapter 10. Springer-Verlag, 1997.

7 V. D. Blondel, O. Bournez, P. Koiran, C. H. Papadimitriou, and J. N. Tsitsiklis. Deciding stability and mortality of piecewise affine dynamical systems. *Theor. Comput. Sci.*, 255(1-2):687–696, 2001.

8 O. Bournez and M. Cosnard. On the computational power of dynamical systems and hybrid systems. *Theor. Comput. Sci.*, 168(2):417–459, 1996.

9 O. Bournez and F. Garnier. Proving positive almost-sure termination. In J. Giesl, editor, *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 2005.

10 O. Bournez and F. Garnier. Proving positive almost sure termination under strategies. In F. Pfenning, editor, *RTA*, volume 4098 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 2006.

11 O. Bournez and M. Hoyrup. Rewriting logic and probabilities. In R. Nieuwenhuis, editor, *RTA*, volume 2706 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2003.

12 M. Brin and G. Stuck. *Introduction to Dynamical Systems*. Cambridge University Press, 2002.

13 J. de Bakker and E. de Vink. *Control Flow Semantics*. The MIT Press, 1996.

14 J.-C. Delvenne. Turing universality in dynamical systems. In A. Beckmann, U. Berger, B. Löwe, and J. V. Tucker, editors, *CiE*, volume 3988 of *Lecture Notes in Computer Science*, pages 147–152. Springer, 2006.

15 J.-C. Delvenne, P. Kurka, and V. D. Blondel. Decidability and universality in symbolic dynamical systems. *Fundamenta Informaticae*, 74(4):463–490, 2006.

16 S. Galatolo, M. Hoyrup, and C. Rojas. Effective symbolic dynamics, random points, statistical behavior, complexity and entropy. *Information and Computation*, 208(1):23–41, 2010.

17 A. Katok and B. Hasselblatt. *Introduction to the Modern Theory of Dynamical Systems*. Cambridge University Press, 1995.

18 R. Kennaway and F.-J. de Vries. Infinitary rewriting. In Terese [27], chapter 12, pages 668–711.

19 R. Kennaway, J. Klop, R. Sleep, and F.-J. de Vries. Transfinite reductions in orthogonal term rewriting. *Information and Computation*, 119(1):18–38, 1995.

20 J. Ketema. *Böhm-Like Trees for Rewriting Systems*. PhD thesis, Vrije Universiteit Amsterdam, 2006.

21 J. Ketema. Reinterpreting compression in infinitary rewriting. In *Proceedings of the 23rd International Conference on Rewriting Techniques and Applications (RTA 2012)*, Leibniz International Proceedings in Informatics, 2012.

22 B. Kitchens. *Symbolic Dynamics: One-Sided, Two-Sided and Countable State Markov Shifts*. Universitext. Springer-Verlag, 1997.

23 B. Kitchens. Symbolic dynamics of tree maps. *Journal of Difference Equations and Applications*, 15(1):71–76, 2009.

24 G. Kreisel, D. Lacombe, and J. Shoenfield. Fonctionelles récursivement définissables et fonctionelles récursives. *Comptes Rendus Hebdomadaires des Séanced de l'Academie des Sciences*, 245:399–402, 1957.

25 D. Lind and B. Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, 1995.

26 J. G. Simonsen. On the computational complexity of the languages of general symbolic dynamical systems and beta-shifts. *Theor. Comput. Sci.*, 410(47-49):4878–4891, 2009.

27 Terese, editor. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

28 P. Walters. *An Introduction to Ergodic Theory*, volume 79 of *Graduate Texts in Mathematics*. Springer-Verlag, 1981.

Infinitary Term Graph Rewriting is Simple, Sound and Complete

Patrick Bahr

Department of Computer Science, University of Copenhagen (DIKU)

Abstract

Based on a simple metric and a simple partial order on term graphs, we develop two infinitary calculi of term graph rewriting. We show that, similarly to infinitary term rewriting, the partial order formalisation yields a conservative extension of the metric formalisation of the calculus. By showing that the resulting calculi simulate the corresponding well-established infinitary calculi of term rewriting in a sound and complete manner, we argue for the appropriateness of our approach to capture the notion of infinitary term graph rewriting.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems, F.1.1 Models of Computation

Keywords and phrases term graphs, partial order, metric, infinitary rewriting, graph rewriting

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.69

Category Regular Research Paper

1 Introduction

Term graph rewriting provides an efficient technique for implementing term rewriting by avoiding duplication of terms and instead relying on pointers in order to refer to a term several times [8]. Due to cycles, *finite* term graphs may represent *infinite* terms, and, correspondingly, *finite* term graph reductions may represent *transfinite* term reductions. Kennaway et al. [16] showed that finite term graph reductions simulate a restricted class of transfinite term reductions, called *rational reductions*, in a sound and complete manner via the unravelling mapping $\mathcal{U}(\cdot)$ from term graphs to terms. More precisely, given a term graph rewriting system $\mathcal{R}$ and a finite term graph g, we have for each finite term graph reduction $g \to^*_{\mathcal{R}} h$, a rational term reduction $\mathcal{U}(g) \twoheadrightarrow_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$ (soundness), and conversely, for each rational term reduction $\mathcal{U}(g) \twoheadrightarrow_{\mathcal{U}(\mathcal{R})} t$, there is a term graph reduction $g \to^*_{\mathcal{R}} h$ and a rational term reduction $t \twoheadrightarrow_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$ (completeness). Since term graph reduction steps may contract several term redexes simultaneously, the completeness result has to be formulated in this weaker form. Note, however, that this completeness property subsumes completeness of normalising reductions: for each rational reduction $\mathcal{U}(g) \twoheadrightarrow_{\mathcal{U}(\mathcal{R})} t$ to a normal form t, there is a reduction $g \to^*_{\mathcal{R}} h$ with $\mathcal{U}(h) = t$.

In this paper, we aim to resolve the asymmetry in the comparison of term rewriting and term graph rewriting by studying transfinite term graph reductions. To this end, we develop two infinitary calculi of term graph rewriting by generalising the notions of strong convergence on terms, based on a metric [15] resp. partial order [4], to term graphs. Instead of the complicated structures that we have used in our previous approach to weak convergence on term graphs [5], we adopt a rather simple and intuitive metric resp. partial order [6].

After summarising the basic theory of infinitary term rewriting (Section 2) and the fundamental concepts concerning term graphs (Section 3), we present a metric and a partial order on term graphs (Section 4). Based on these two structures, we define the notions of strong m-convergence resp. strong p-convergence and show that – akin to term rewriting – both coincide on total term graphs and that strong p-convergence is normalising (Section 5).

In Section 6, we present the main result of this paper: strongly p-converging term graph reductions are sound and complete w.r.t. strongly p-converging term reductions in the sense of Kennaway et al. [16] explained above.

This result comes with some surprise, though, as Kennaway et al. [16] argued that infinitary term graph rewriting cannot adequately simulate infinitary term rewriting. In particular, they present a counterexample for the completeness of an informally defined infinitary calculus of term graph rewriting. This counterexample indeed shows that strongly m-converging term graph reductions are not complete for strongly m-converging term reductions.

However, using the correspondence between strong p-convergence and m-convergence, we can derive soundness of the metric calculus from the soundness of the partial order calculus. Moreover, we prove that the metric calculus is still complete for *normalising* reductions. We thus argue that strong m-convergence, too, can be adequately simulated by term graph rewriting. In fact, in their original work on term graph rewriting [8], Barendregt et al. showed completeness only for normalising reductions in order to argue for the adequacy of acyclic finite term graph rewriting for simulating finite term rewriting.

Due to space restrictions, we could not include all proofs in the main body of this paper. All missing proofs can be found in the companion report [7].

2 Infinitary Term Rewriting

We assume familiarity with the basic theory of term rewriting [19], ordinal numbers, orders and topological spaces [14]. Below, we give an outline of infinitary term rewriting [15, 4].

We denote ordinal numbers by lower case Greek letters $\alpha, \beta, \gamma, \lambda, \iota$. A *sequence* S of length α in a set A, written $(a_\iota)_{\iota<\alpha}$, is a function from α to A with $\iota \mapsto a_\iota$ for all $\iota \in \alpha$. We write $|S|$ for the length α of S. If α is a limit ordinal, S is called *open*; otherwise it is called *closed*. Given two sequences S, T, we write $S \cdot T$ to denote their concatenation and $S \leq T$ (resp. $S < T$) if S is a (proper) prefix of T. The prefix of T of length $\beta \leq |T|$ is denoted $T|_\beta$. For a set A, we write A^* to denote the set of finite sequences over A. For a finite sequence $(a_i)_{i<n} \in A^*$, we also write $\langle a_0, a_1, \dots, a_{n-1}\rangle$. In particular, $\langle\rangle$ denotes the empty sequence.

We consider the sets $\mathcal{T}^\infty(\Sigma)$ and $\mathcal{T}(\Sigma)$ of (possibly infinite) *terms* resp. *finite terms* over a signature Σ. Each symbol f has an associated arity $\mathsf{ar}(f)$, and we write $\Sigma^{(n)}$ for the set of symbols in Σ with arity n. For rewrite rules, we consider the signature $\Sigma_\mathcal{V} = \Sigma \uplus \mathcal{V}$ that extends the signature Σ with a set $\mathcal{V}$ of nullary variable symbols. For terms $s, t \in \mathcal{T}^\infty(\Sigma)$ and a position $\pi \in \mathcal{P}(t)$ in t, we write $t|_\pi$ for the *subterm* of t at π, $t(\pi)$ for the symbol in t at π, and $t[s]_\pi$ for the term t with the subterm at π replaced by s.

A *term rewriting system* (TRS) $\mathcal{R}$ is a pair (Σ, R) consisting of a signature Σ and a set R of *term rewrite rules* of the form $l \to r$ with $l \in \mathcal{T}^\infty(\Sigma_\mathcal{V}) \setminus \mathcal{V}$ and $r \in \mathcal{T}^\infty(\Sigma_\mathcal{V})$ such that all variables occurring in r also occur in l. If the left-hand side of each rule in a TRS $\mathcal{R}$ is finite, then $\mathcal{R}$ is called *left-finite*. Every TRS $\mathcal{R}$ defines a rewrite relation $\to_\mathcal{R}$ as usual: $s \to_\mathcal{R} t$ iff there is a position $\pi \in \mathcal{P}(s)$, a rule $\rho\colon\ l \to r \in R$, and a substitution σ such that $s|_\pi = l\sigma$ and $t = s[r\sigma]_\pi$. We write $s \to_{\pi,\rho} t$ in order to indicate the applied rule ρ and the position π. The subterm $s|_\pi$ is called a *redex* and is said to be *contracted* to $r\sigma$.

The metric $\mathbf{d}$ on $\mathcal{T}^\infty(\Sigma)$ that is used in the setting of infinitary term rewriting is defined by $\mathbf{d}(s,t) = 0$ if $s = t$ and $\mathbf{d}(s,t) = 2^{-k}$ if $s \neq t$, where k is the minimal depth at which s and t differ. The pair $(\mathcal{T}^\infty(\Sigma), \mathbf{d})$ is known to form a *complete ultrametric space* [2].

A *reduction* in a term rewriting system $\mathcal{R}$, is a sequence $S = (t_\iota \to_{\pi_\iota} t_{\iota+1})_{\iota<\alpha}$ of reduction steps in $\mathcal{R}$. The reduction S is called strongly *m-continuous* if $\lim_{\iota\to\lambda} t_\iota = t_\lambda$ and the depths of contracted redexes $(|\pi_\iota|)_{\iota<\lambda}$ tend to infinity, for each limit ordinal $\lambda < \alpha$. A reduction S is said to strongly m-converge to t, written $S\colon t_0 \overset{m}{\twoheadrightarrow}_{\mathcal{R}} t$, if it is strongly m-continuous and either S is closed with $t = t_\alpha$ or S is open with $t = \lim_{\iota\to\alpha} t_\iota$ and the depths of contracted redexes $(|\pi_\iota|)_{\iota<\alpha}$ tend to infinity.

▶ **Example 2.1.** Consider the rule $\rho\colon Y\,x \to x\,(Y\,x)$ defining the fixed point combinator Y in an applicative language. If we use an explicit function symbol @ instead of juxtaposition to denote application, ρ reads $@(Y, x) \to @(x, @(Y, x))$. Given a term t, we get the reduction

$$S\colon\ Y\,t \to_\rho t\,(Y\,t) \to_\rho t\,(t\,(Y\,t)) \to_\rho t\,(t\,(t\,(Y\,t))) \to_\rho \dots$$

which strongly m-converges to the infinite term $t\,(t\,(\dots))$.

As another example, consider the rule $\rho'\colon\ f(x) \to f(g(x))$ and its induced reduction

$$T\colon\ h(c, f(c)) \to_{\rho'} h(c, f(g(c))) \to_{\rho'} h(c, f(g(g(c)))) \to h(c, f(g(g(g(c))))) \to_{\rho'} \dots$$

Although the underlying sequence of terms converges in the metric space $(\mathcal{T}^\infty(\Sigma), \mathbf{d})$, viz. to the infinite term $h(c, f(g(g(\dots))))$, the reduction T does not strongly m-converges since the depth of the contracted redexes does not tend to infinity but instead stays at 1.

The partial order $\le_\bot$ is defined on *partial terms*, i.e. terms over signature $\Sigma_\bot = \Sigma \uplus \{\bot\}$, with $\bot$ a nullary symbol. It is characterised as follows: $s \le_\bot t$ iff t can be obtained from s by replacing each occurrence of $\bot$ by some partial term. The pair $(\mathcal{T}^\infty(\Sigma_\bot), \le_\bot)$ forms a *complete semilattice* [13]. A partially ordered set $(A, \le)$ is called a *complete partial order* (*cpo*) if it has a *least element* and every *directed subset* D of A has a *least upper bound* (*lub*) $\bigsqcup D$ in A. If, additionally, every *non-empty* subset B of A has a *greatest lower bound* (*glb*) $\bigsqcap B$, then $(A, \le)$ is called a *complete semilattice*. This means that for complete semilattices the *limit inferior* $\liminf_{\iota\to\alpha} a_\iota = \bigsqcup_{\beta<\alpha} \left(\bigsqcap_{\beta\le\iota<\alpha} a_\iota\right)$ of a sequence $(a_\iota)_{\iota<\alpha}$ is always defined.

In the partial order approach to infinitary rewriting, convergence is defined by the limit inferior. Since we are considering strong convergence, the positions π_ι at which reductions take place are taken into consideration as well. In particular, we consider, for each reduction step $t_\iota \to_{\pi_\iota} t_{\iota+1}$ at position π_ι, the *reduction context* $c_\iota = t_\iota[\bot]_{\pi_\iota}$, i.e. the starting term with the redex at π_ι replaced by $\bot$. To indicate the reduction context c_ι of a reduction step, we also write $t_\iota \to_{c_\iota} t_{\iota+1}$. A reduction $S = (t_\iota \to_{c_\iota} t_{\iota+1})_{\iota<\alpha}$ is called *strongly p-continuous* if $\liminf_{\iota<\lambda} c_\iota = t_\lambda$ for each limit ordinal $\lambda < \alpha$. The reduction S is said to *strongly p-converge* to a term t, written $S\colon t_0 \overset{p}{\twoheadrightarrow}_{\mathcal{R}} t$, if it is strongly p-continuous and either S is closed with $t = t_\alpha$, or S is open with $\liminf_{\iota<\alpha} c_\iota = t$. If $S\colon t_0 \overset{p}{\twoheadrightarrow}_{\mathcal{R}} t$ and t as well as all t_ι with $\iota < \alpha$ are total, i.e. contained in $\mathcal{T}^\infty(\Sigma)$, then we say that S strongly p-converges to t in $\mathcal{T}^\infty(\Sigma)$.

The distinguishing feature of the partial order approach is that, since the partial order on terms forms a complete semilattice, each continuous reduction also converges. It provides a conservative extension to strong m-convergence that allows rewriting modulo *meaningless terms* [4] by rewriting terms to $\bot$ if they are divergent according to the metric calculus.

▶ **Example 2.2.** Reconsider S and T from Example 2.1. S has the same convergence behaviour in the partial order setting, viz. $S\colon Y\,t \overset{p}{\twoheadrightarrow} t\,(t\,(\dots))$. However, while the reduction T does not strongly m-converge, it does strongly p-converge, viz. $T\colon\ h(c, f(c)) \overset{p}{\twoheadrightarrow} h(c, \bot)$.

The relation between m- and p-convergence illustrated in the examples above is characteristic: strong p-convergence is a conservative extension of strong m-convergence.

▶ **Theorem 2.3** ([4]). *For every reduction S in a TRS the following equivalence holds:*

$$S\colon\ s \overset{m}{\twoheadrightarrow}_{\mathcal{R}} t \qquad \textit{iff} \qquad S\colon\ s \overset{p}{\twoheadrightarrow}_{\mathcal{R}} t \text{ in } \mathcal{T}^\infty(\Sigma).$$

In the remainder of this paper, we shall develop a generalisation of both strong m- and p-convergence to term graphs that maintains the above correspondence, and additionally simulates term reductions in a sound and complete way.

3 Graphs and Term Graphs

The notion of term graphs that we employ in this paper is taken from Barendregt et al. [8].

▶ **Definition 3.1** (graphs)**.** Let Σ be a signature. A *graph* over Σ is a tuple $g = (N, \mathsf{lab}, \mathsf{suc})$ 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. If $\mathsf{suc}(n) = \langle n_0, \dots, n_{k-1}\rangle$, then we write $\mathsf{suc}_i(n)$ for n_i. Moreover, we use the abbreviation $\mathsf{ar}_g(n)$ for the arity $\mathsf{ar}(\mathsf{lab}(n))$ of n in g.

▶ **Definition 3.2** (paths, reachability)**.** Let $g = (N, \mathsf{lab}, \mathsf{suc})$ be a graph and $n, m \in N$. A *path* in g from n to m is a finite sequence $\pi \in \mathbb{N}^*$ such that either π is empty and $n = m$, or $\pi = \langle i\rangle \cdot \pi'$ with $0 \le i < \mathsf{ar}_g(n)$ and the suffix π' is a path in g from $\mathsf{suc}_i(n)$ to m. If there exists a path from n to m in g, we say that m is *reachable* from n in g.

▶ **Definition 3.3** (term graphs)**.** 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 class of all term graphs over Σ is denoted $\mathcal{G}^\infty(\Sigma)$. 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, which consists of all nodes reachable from n in h.

Paths in a graph are not absolute but relative to a starting node. In term graphs, however, we have a distinguished root node from which each node is reachable. Paths relative to the root node are central for dealing with term graphs modulo isomorphism:

▶ **Definition 3.4** (positions, depth, trees)**.** Let $g \in \mathcal{G}^\infty(\Sigma)$ and $n \in N^g$. A *position* of n in g 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 $\mathcal{P}_g(n)$. A position $\pi \in \mathcal{P}_g(n)$ is called *minimal* if no proper prefix $\pi' < \pi$ is in $\mathcal{P}_g(n)$. The set of all minimal positions of n in g is denoted $\mathcal{P}^m_g(n)$. The *depth* of n in g, denoted $\mathsf{depth}_g(n)$, is the minimum of the lengths of the positions of n in g. For a position $\pi \in \mathcal{P}(g)$, we write $\mathsf{node}_g(\pi)$ for the unique node $n \in N^g$ with $\pi \in \mathcal{P}_g(n)$, $g(\pi)$ for its symbol $\mathsf{lab}^g(n)$, and $g|_\pi$ for the sub-term graph $g|_n$. The term graph g is called a *term tree* if each node in g has exactly one position.

Note that the labelling function of graphs – and thus term graphs – is *total*. In contrast, Barendregt et al. [8] considered *open* (term) graphs with a *partial* labelling function such that unlabelled nodes denote holes or variables. This partiality is reflected in their notion of homomorphisms in which the homomorphism condition is suspended for unlabelled nodes.

Instead of a partial node labelling function, we chose a *syntactic* approach that is more flexible and closer to the representation in terms. Variables, holes and "bottoms" are labelled by a distinguished set of constant symbols and the notion of homomorphisms is parametrised by a set of constant symbols Δ for which the homomorphism condition is suspended:

▶ **Definition 3.5** (Δ-homomorphisms)**.** Let Σ be a signature, $\Delta \subseteq \Sigma^{(0)}$, and $g, h \in \mathcal{G}^\infty(\Sigma)$. A function $\phi\colon N^g \to N^h$ is called *homomorphic* in $n \in N^g$ if the following holds:

$$\mathsf{lab}^g(n) = \mathsf{lab}^h(\phi(n)) \qquad \text{(labelling)}$$

$$\phi(\mathsf{suc}^g_i(n)) = \mathsf{suc}^h_i(\phi(n)) \quad \text{for all } 0 \le i < \mathsf{ar}_g(n) \qquad \text{(successor)}$$

A Δ-*homomorphism* ϕ from g to h, denoted $\phi\colon g \to_\Delta h$, is a function $\phi\colon N^g \to N^h$ that is homomorphic in n for all $n \in N^g$ with $\mathsf{lab}^g(n) \notin \Delta$ and satisfies $\phi(r^g) = r^h$.

Note that, in contrast to Barendregt et al. [8], we require that root nodes are mapped to root nodes. This additional requirement makes our generalised notion of homomorphisms more akin to that of Barendsen [9]: for $\Delta = \emptyset$, we obtain his notion of homomorphisms.

Nodes labelled with a symbol from Δ can be thought of as holes in the term graphs, which can be filled with other term graphs. For example, if we have a distinguished set of variable symbols $\mathcal{V} \subseteq \Sigma^{(0)}$, we can use $\mathcal{V}$-homomorphisms to formalise the matching of a term graph against a term graph rule, which requires the instantiation of variables.

Note that Δ-homomorphisms are unique [5], i.e. there is at most one Δ-homomorphism from one term graph to another. Consequently, whenever there are two Δ-homomorphisms $\phi\colon g \to_\Delta h$ and $\psi\colon h \to_\Delta g$, they are inverses of each other, i.e. Δ-*isomorphisms*. If two term graphs are Δ-*isomorphic*, we write $g \cong_\Delta h$.

For the two special cases $\Delta = \emptyset$ and $\Delta = \{\sigma\}$, we write $\phi\colon g \to h$ resp. $\phi\colon g \to_\sigma h$ instead of $\phi\colon g \to_\Delta h$ and call ϕ a homomorphism resp. a σ-homomorphism. The same convention applies to Δ-isomorphisms.

Since we are studying modes of convergence over term graphs, we want to reason modulo isomorphism. The following notion of canonical term graphs will allow us to do that:

▶ **Definition 3.6** (canonical term graphs)**.** A term graph g is called *canonical* if $n = \mathcal{P}_g(n)$ for each $n \in N^g$. The set of all canonical term graphs over Σ is denoted $\mathcal{G}^\infty_\mathcal{C}(\Sigma)$.

For each term graph g, we can give a unique canonical term graph $\mathcal{C}(g)$ isomorphic to g:

$$N^{\mathcal{C}(g)} = \{\mathcal{P}_g(n) \,|\, n \in N\} \qquad r^{\mathcal{C}(g)} = \mathcal{P}_g(r)$$
$$\mathsf{lab}^{\mathcal{C}(g)}(\mathcal{P}_g(n)) = \mathsf{lab}(n) \qquad \mathsf{suc}_i^{\mathcal{C}(g)}(\mathcal{P}_g(n)) = \mathcal{P}_g(\mathsf{suc}_i(n)) \quad \text{for all } n \in N, 0 \leq i < \mathsf{ar}_g(n)$$

As we have shown previously [5], this indeed yields a canonical representation of term graphs, viz. $g \cong h$ iff $\mathcal{C}(g) = \mathcal{C}(h)$ for all term graphs g, h.

Note that the set of nodes $N^{\mathcal{C}(g)}$ above forms a partition of the set of positions in g. We write $\sim_g$ for the equivalence relation on $\mathcal{P}(g)$ that is induced by this partition. That is, $\pi_1 \sim_g \pi_2$ iff $\mathsf{node}_g(\pi_1) = \mathsf{node}_g(\pi_2)$. The structure of a term graph g is uniquely determined by its set of positions $\mathcal{P}(g)$, the labelling $g(\cdot)\colon \pi \mapsto g(\pi)$, and the equivalence $\sim_g$. We will call such a triple $(\mathcal{P}(g), g(\cdot), \sim_g)$ a *labelled quotient tree*. Labelled quotient trees uniquely represent term graphs up to isomorphism. In other words: labelled quotient trees uniquely represent canonical term graphs. For a more axiomatic treatment of labelled quotient tree that studies these relationships, we refer to our previous work [5].

We can characterise Δ-homomorphisms in terms of labelled quotient trees:

▶ **Lemma 3.7** ([5])**.** *Given $g, h \in \mathcal{G}^\infty(\Sigma)$, there is a $\phi\colon g \to_\Delta h$ iff for all $\pi, \pi' \in \mathcal{P}(g)$,*

$$\text{(a) } \pi \sim_g \pi' \quad \Longrightarrow \quad \pi \sim_h \pi', \text{ and} \qquad \text{(b) } g(\pi) = h(\pi) \quad \text{whenever} \quad g(\pi) \notin \Delta.$$

Intuitively, (a) means that h has at least as much sharing of nodes as g has, whereas (b) means that h has at least the same non-Δ-symbols as g.

Given a term tree g, the equivalence $\sim_g$ is the identity relation $\mathcal{I}_{\mathcal{P}(g)}$ on $\mathcal{P}(g)$, i.e. $\pi_1 \sim_g \pi_2$ iff $\pi_1 = \pi_2$. There is an obvious one-to-one correspondence between canonical term *trees* and terms: a term $t \in \mathcal{T}^\infty(\Sigma)$ corresponds to the canonical term tree given by the labelled quotient tree $(\mathcal{P}(t), t(\cdot), \mathcal{I}_{\mathcal{P}(t)})$. We thus consider the set of terms $\mathcal{T}^\infty(\Sigma)$ as the subset of term trees in $\mathcal{G}^\infty_\mathcal{C}(\Sigma)$.

With this correspondence in mind, we define the *unravelling* of a term graph g, denoted $\mathcal{U}(g)$, as the unique term t such that there is a homomorphism $\phi\colon t \to g$.

▶ **Example 3.8.** Consider the term graphs g_2 and h_0 illustrated in Figure 1. The unravelling of g_2 is the term $@(f, @(f, @(Y, f)))$ whereas the unravelling of the cyclic term graph h_0 is the infinite term $@(f, @(f, \dots))$.

4 Two Simple Modes of Convergence for Term Graphs

In a previous attempt to generalise the modes of convergence of term rewriting to term graphs, we developed a metric and a partial order on term graphs that were both rather complicated [5]. While the resulting notions of weak convergence have a correspondence similar to that for terms (cf. Theorem 2.3), they are also limited as we explain below. In this paper, we shall use a much simpler and more intuitive approach that we recently developed [6], and which we summarise briefly below.

Like for terms, we move to a signature $\Sigma_\perp = \Sigma \uplus \{\perp\}$ to define a partial order on term graphs. Term graphs over signature $\Sigma_\perp$ are also referred to as *partial* whereas term graphs over Σ are referred to as *total*. In order to generalise the partial order $\leq_\perp$ on terms to term graphs, we make use of the observation that $\perp$-homomorphisms characterise the partial order $\leq_\perp$: given two terms $s, t \in \mathcal{T}^\infty(\Sigma_\perp)$, we have $s \leq_\perp t$ iff there is a $\perp$-homomorphism $\phi\colon s \to_\perp t$. In our previous work, we have used a restricted form of $\perp$-homomorphisms in order to define a partial order on term graphs [5]. In this paper, however, we simply take $\perp$-homomorphism as the definition of the partial order on term graphs. The *simple partial order* $\leq^{\mathsf{S}}_\perp$ on $\mathcal{G}^\infty_\mathcal{C}(\Sigma_\perp)$ is defined as follows: $g \leq^{\mathsf{S}}_\perp h$ iff there is a $\perp$-homomorphism $\phi\colon s \to_\perp t$. Hence, we get the following characterisation, according to Lemma 3.7:

▶ **Corollary 4.1.** *Let $g, h \in \mathcal{G}^\infty_\mathcal{C}(\Sigma_\perp)$. Then $g \leq^{\mathsf{S}}_\perp h$ iff, for all $\pi, \pi' \in \mathcal{P}(g)$, we have*

$$\text{(a) } \pi \sim_g \pi' \quad \Longrightarrow \quad \pi \sim_h \pi' \qquad \text{(b) } g(\pi) = h(\pi) \quad \text{if } g(\pi) \in \Sigma.$$

With this partial order on term graphs, we indeed get a complete semilattice:

▶ **Theorem 4.2** ([6]). *The pair $(\mathcal{G}^\infty_\mathcal{C}(\Sigma_\perp), \leq^{\mathsf{S}}_\perp)$ forms a complete semilattice. In particular, the limit inferior of a sequence $(g_\iota)_{\iota<\alpha}$ is given by the labelled quotient tree $(P, \sim, l)$:*

$$P = \bigcup\nolimits_{\beta<\alpha} \{\pi \in \mathcal{P}(g_\beta) \,|\, \forall \pi' < \pi \forall \beta \leq \iota < \alpha\colon g_\iota(\pi') = g_\beta(\pi')\}$$

$$\sim \; = (P \times P) \cap \bigcup\nolimits_{\beta<\alpha} \bigcap\nolimits_{\beta\leq\iota<\alpha} \sim_{g_\iota}$$

$$l(\pi) = \begin{cases} g_\beta(\pi) & \text{if } \exists\beta < \alpha \forall \beta \leq \iota < \alpha\colon g_\iota(\pi) = g_\beta(\pi) \\ \perp & \text{otherwise} \end{cases} \qquad \text{for all } \pi \in P$$

In order to generalise the metric $\mathbf{d}$ on terms to term graphs, we need to formalise what it means for two term graphs to be "equal" up to a certain depth. To this end, we define for each term graph $g \in \mathcal{G}^\infty(\Sigma_\perp)$ and $d \in \mathbb{N}$ the *simple truncation* $g{\dagger}d$ as the term graph obtained from g by relabelling each node at depth d with $\perp$ and (thus) removing all nodes at depth greater than d. The distance $\mathbf{d}_\dagger(g, h)$ between two term graphs $g, h \in \mathcal{G}^\infty(\Sigma)$ is then defined as 0 if $g \cong h$ and otherwise as 2^{-d} with d the greatest $d \in \mathbb{N}$ with $g{\dagger}d \cong h{\dagger}d$. This definition indeed yields a complete ultrametric space:

▶ **Theorem 4.3** ([6]). *The pair $(\mathcal{G}^\infty_\mathcal{C}(\Sigma), \mathbf{d}_\dagger)$ forms a complete ultrametric space. In particular, the limit of each Cauchy sequence $(g_\iota)_{\iota<\alpha}$ is given by the labelled quotient tree $(P, l, \sim)$:*

$$P = \liminf_{\iota\to\alpha} \mathcal{P}(g_\iota) = \bigcup_{\beta<\alpha} \bigcap_{\beta\leq\iota<\alpha} \mathcal{P}(g_\iota) \qquad \sim \; = \liminf_{\iota\to\alpha} \sim_{g_\iota} = \bigcup_{\beta<\alpha} \bigcap_{\beta\leq\iota<\alpha} \sim_{g_\iota}$$

$$l(\pi) = g_\beta(\pi) \quad \text{for some } \beta < \alpha \text{ with } g_\iota(\pi) = g_\beta(\pi) \text{ for each } \beta \leq \iota < \alpha \qquad \text{for all } \pi \in P$$

The metric space that we have previously studied [5] was similarly defined in terms of a truncation. However, we used a much more complicated notion of truncation that would retain certain nodes of depth greater than d.

Similarly to the corresponding modes of convergence on terms, we have that if a sequence of total term graphs $(g_\iota)_{\iota<\alpha}$ converges in the metric space $(\mathcal{G}^\infty_\mathcal{C}(\Sigma_\bot), \mathbf{d}_\dagger)$, then $\lim_{\iota\to\alpha} g_\iota = \liminf_{\iota\to\alpha} g_\iota$. However, unlike in the setting of terms, the converse is not true! That is, if $\liminf_{\iota\to\alpha} g_\iota$ is a total term graph, then it is not necessarily equal to $\lim_{\iota\to\alpha} g_\iota$ – in fact, $(g_\iota)_{\iota<\alpha}$ might not even converge at all. As a consequence, we are not able to obtain a correspondence in the vein of Theorem 2.3 for weak convergence. In the next section, we will show that we do, however, obtain such a correspondence for strong convergence.

Note that the more restrictive partial order and metric space that we have studied in our previous work [5] does yield the above described correspondence for weak convergence. However, this result comes at the expense of generality and intuition: the convergence behaviour illustrated in Figure 1c, which is intuitively expected and also captured by the partial order $\leq^{\mathsf{S}}_\bot$ and the metric $\mathbf{d}_\dagger$, is not possible in these more restrictive structures [6].

5 Infinitary Term Graph Rewriting

In this paper, we adopt the term graph rewriting framework of Barendregt et al. [8]. In order to represent placeholders in rewrite rules, this framework uses variables – in a manner much 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 set $\mathcal{V}$ of nullary variable symbols.

▶ **Definition 5.1** (term graph rewriting systems).

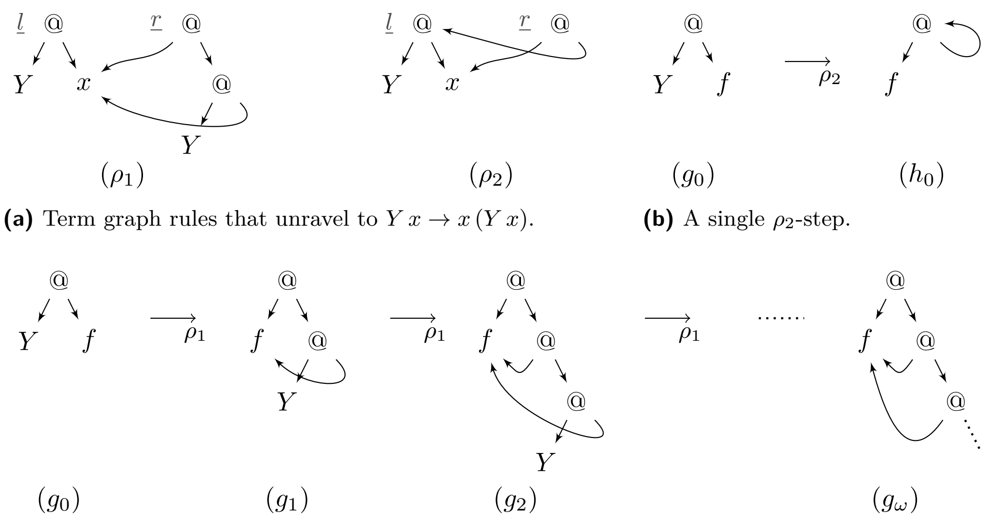
(i) Given a signature Σ, a *term graph rule* ρ over Σ is a triple (g, l, r) where g is a graph over $\Sigma_\mathcal{V}$ and $l, r \in N^g$ such that all nodes in g are reachable from l or r. We write ρ_l resp. ρ_r to denote the left- resp. right-hand side of ρ, i.e. the term graph $g|_l$ resp. $g|_r$. 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 but still reachable from l.

(ii) A *term graph rewriting system (GRS)* $\mathcal{R}$ is a pair (Σ, R) with Σ a signature and R a set of term graph rules over Σ.

The notion of unravelling straightforwardly extends to term graph rules: let ρ be a term graph rule with ρ_l and ρ_r its left- resp. right-hand side term graph. The *unravelling* of 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 $(\Sigma, \{\mathcal{U}(\rho) \mid \rho \in R\})$.

▶ **Example 5.2.** Figure 1a shows two term graph rules which both unravel to the term rule $\rho\colon Y\,x \to x\,(Y\,x)$ from Example 2.1. Note that sharing of nodes is used both to refer to variables in the left-hand side from the right-hand side, and in order to simulate duplication.

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 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 the sub-term graph in g rooted in n, the *redex*. The $\mathcal{V}$-homomorphism ϕ allows us to instantiate 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



(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.

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 above construction, this induces a *pre-reduction step* $\psi\colon\ g \mapsto_{n,\rho,n'} h$ from g to h. In order to indicate the underlying GRS $\mathcal{R}$, we also write $\psi\colon\ g \mapsto_{\mathcal{R}} h$.

The definition of term graph rewriting in the form of pre-reduction steps is very operational in style. The result of applying a rewrite rule to a term graph is constructed in several steps by manipulating nodes and edges explicitly. While this is beneficial for implementing a rewriting system, it is problematic for reasoning on term graphs modulo isomorphism, which is necessary for introducing notions of convergence. In our case, however, this does not cause any harm since the construction of the result term graph of a pre-reduction step is invariant under isomorphism. This observation justifies the following definition of reduction steps:

▶ **Definition 5.3.** 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 $m = \mathcal{P}_{h'}(m')$. Similarly to pre-reduction steps, we write $\phi\colon\ g \to_\mathcal{R} h$ or $\phi\colon\ g \to h$ for short.

In other words, a reduction step is a canonicalised pre-reduction step. Figures 1b and 1c show various (pre-)reduction steps derived from the rules in Figure 1a.

5.1 Reduction Contexts

The idea of strong convergence is to conservatively approximate the convergence behaviour somewhat independently from the actual rules that are applied. Strong m-convergence in TRSs requires that the depth of the redexes tends to infinity thereby assuming that anything at the depth of the redex or below is potentially affected by a reduction step. Strong p-convergence, on the other hand, uses a better approximation that only assumes that the redex is affected by a reduction step – not however other subterms at the same depth. To this end strong p-convergence uses a notion of reduction contexts – essentially the term minus the redex – for the formation of limits. In this section, we shall devise a corresponding notion of reduction contexts on term graphs and argue for its adequacy for formalising strong p-convergence. The following definition provides the basic construction that we shall use:

▶ **Definition 5.4.** Let $g \in \mathcal{G}^\infty(\Sigma_\perp)$ and $n \in N^g$. The *local truncation* of g at n, denoted $g\backslash n$, is obtained from g by labelling n with $\perp$ and removing all outgoing edges from n as well as all nodes that thus become unreachable from the root.

▶ **Lemma 5.5.** *For each $g \in \mathcal{G}^\infty(\Sigma_\perp)$ and $n \in N^g$, the local truncation $g\backslash n$ has the following labelled quotient tree $(P, l, \sim)$:*

$$P = \{\pi \in \mathcal{P}(g) \mid \forall \pi' < \pi\colon \pi' \notin \mathcal{P}_g(n)\} \qquad l(\pi) = \begin{cases} g(\pi) & \text{if } \pi \notin \mathcal{P}_g(n) \\ \perp & \text{if } \pi \in \mathcal{P}_g(n) \end{cases} \quad \text{for all } \pi \in P$$
$$\sim \,=\, \sim_g \cap\, P \times P$$

As a corollary of Lemma 5.5 and Corollary 4.1 we obtain the following:

▶ **Corollary 5.6.** *For each $g \in \mathcal{G}^\infty(\Sigma_\perp)$ and $n \in N^g$, we have $g\backslash n \leq_\perp^{\mathsf{S}} g$.*

It is also possible – although cumbersome – to show that, given a reduction step $g \to_n h$ at node n, the local truncation $g\backslash n$ is isomorphic to the term graph that is obtained from h by essentially relabelling the positions $\mathcal{P}_g(n)$ occurring in h with $\perp$. For this term graph, denoted $h\backslash[\mathcal{P}_g(n)]$, we then also have $h\backslash[\mathcal{P}_g(n)] \leq_\perp^{\mathsf{S}} h$. By combining this with Corollary 5.6, we eventually obtain the following fundamental property of reduction contexts:

▶ **Proposition 5.7.** *Given a reduction step $g \to_n h$, we have $g\backslash n \leq_\perp^{\mathsf{S}} g, h$.*

This means that the local truncation at the root of the redex is preserved by reduction steps and is therefore an adequate notion of reduction context for strong p-convergence [3].

5.2 Strong Convergence

Now that we have an adequate notion of reduction contexts, we can define strong p-convergence on term graphs analogously to strong p-convergence on terms. For strong m-convergence, we simply take the same notion of depth that we already used for the definition of the simple truncation $g\dagger d$ and thus the simple metric $\mathbf{d}_\dagger$.

▶ **Definition 5.8.** 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 of a graph reduction step.

(ii) Let $S = (g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\alpha}$ be a reduction in $\mathcal{R}$. S is *strongly 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 *strongly m-converges* to g in $\mathcal{R}$, denoted $S\colon g_0 \overset{m}{\twoheadrightarrow}_{\mathcal{R}} g$, if it is strongly 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.

(iii) Let $S = (g_\iota \to_{c_\iota} g_{\iota+1})_{\iota<\alpha}$ be a reduction in $\mathcal{R}_\perp = (\Sigma_\perp, R)$. S is *strongly p-continuous* in $\mathcal{R}$ if $\liminf_{\iota\to\lambda} c_\iota = g_\lambda$ for each limit ordinal $\lambda < \alpha$. S *strongly p-converges* to g in $\mathcal{R}$, denoted $S\colon g_0 \overset{p}{\twoheadrightarrow}_{\mathcal{R}} g$, if it is strongly 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 $\Sigma_\perp$ for the definition of strong p-convergence. However, we can obtain the total fragment of strong p-convergence if we restrict ourselves to total term graphs: a reduction $(g_\iota \to_{\mathcal{R}_\perp} g_{\iota+1})_{\iota<\alpha}$ strongly p-converging to g is called strongly p-converging to g in $\mathcal{G}_{\mathcal{C}}^\infty(\Sigma)$ if g as well as each g_ι is total, i.e. an element of $\mathcal{G}_{\mathcal{C}}^\infty(\Sigma)$.

▶ **Example 5.9.** Figure 1c illustrates an infinite reduction derived from the rule ρ_1 in Figure 1a. Note that the reduction rule is applied to sub-term graphs at increasingly large depth. Since additionally, $g_i\dagger(i+1) \cong g_\omega\dagger(i+1)$ for all $i < \omega$, i.e. $\lim_{i\to\omega} g_i = g_\omega$, the reduction strongly m-converges to the term graph g_ω. Moreover, since each node in g_ω eventually appears in a reduction context and remains stable afterwards, we have $\liminf_{i\to\omega} g_\iota = g_\omega$. Consequently, the reduction also strongly p-converges to g_ω.

The rest of this section is concerned with proving that the above correspondence in convergence behaviour – similarly to infinitary term rewriting (cf. Theorem 2.3) – is characteristic: strong p-convergence in $\mathcal{G}^\infty_\mathcal{C}(\Sigma)$ coincides with strong m-convergence.

Since the partial order $\leq^\mathsf{S}_\perp$ forms a complete semilattice on $\mathcal{G}^\infty_\mathcal{C}(\Sigma_\perp)$ according to Theorem 4.2, we know that strong p-continuity coincides with strong p-convergence:

▶ **Proposition 5.10.** *Each strongly p-continuous reduction in a GRS is strongly p-convergent.*

The two lemmas below form the central properties that link strong m- and p-convergence:

▶ **Lemma 5.11.** *Let $(g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\alpha}$ be an open reduction in a GRS $\mathcal{R}_\perp$. If S strongly p-converges to a total term graph, then $(\mathsf{depth}_{g_\iota}(n_\iota))_{\iota<\alpha}$ tends to infinity.*

▶ **Lemma 5.12.** *Let $(g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\alpha}$ be an open reduction strongly p-converging to g in a GRS $\mathcal{R}_\perp$. If $(g_\iota)_{\iota<\alpha}$ is Cauchy and $(\mathsf{depth}_{g_\iota}(n_\iota))_{\iota<\alpha}$ tends to infinity, then $g \cong \lim_{\iota\to\alpha} g_\iota$.*

The following property, which relates strong m-convergence and -continuity, follows from the fact that our definition of strong m-convergence on term graphs instantiates the abstract notion of strong m-convergence from our previous work [3]:

▶ **Lemma 5.13.** *Let $S = (g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\alpha}$ be an open strongly m-continuous reduction in a GRS. If $(\mathsf{depth}_{g_\iota}(n_\iota))_{\iota<\alpha}$ tends to infinity, then S is strongly m-convergent.*

Proof. Special case of Proposition 5.5 from [3]; cf. [10, Thm. B.2.5] for the correct proof. ◀

Now we have everything in place to prove that strong p-convergence conservatively extends strong m-convergence.

▶ **Theorem 5.14.** *Let $\mathcal{R}$ be a GRS and S a reduction in $\mathcal{R}_\perp$. We then have that*

$$S\colon g \overset{m}{\twoheadrightarrow}_\mathcal{R} h \qquad \textit{iff} \qquad S\colon g \overset{p}{\twoheadrightarrow}_\mathcal{R} h \textit{ in } \mathcal{G}^\infty_\mathcal{C}(\Sigma).$$

Proof. Let $S = (g_\iota \to_{n_\iota} g_{\iota+1})_{\iota<\alpha}$ be a reduction in $\mathcal{R}_\perp$. We prove the "only if" direction by induction on α. The case $\alpha = 0$ is trivial. If α is a successor ordinal, then the statement follows immediately from the induction hypothesis.

Let α be a limit ordinal. As $S\colon g \overset{m}{\twoheadrightarrow}_\mathcal{R} g_\alpha$, we know that $S|_\gamma\colon g \overset{m}{\twoheadrightarrow}_\mathcal{R} g_\gamma$ for all $\gamma < \alpha$. Hence, we can apply the induction hypothesis to obtain that $S|_\gamma\colon g \overset{p}{\twoheadrightarrow}_\mathcal{R} g_\gamma$ for each $\gamma < \alpha$. Thus, S is strongly p-continuous, which means, by Proposition 5.10, that S strongly p-converges to some term graph h'. As S strongly m-converges, we know that $(g_\iota)_{\iota<\alpha}$ is Cauchy and that $(\mathsf{depth}_{g_\iota}(n_\iota))_{\iota<\alpha}$ tends to infinity. Hence, we can apply Lemma 5.12 to obtain that $h' = \lim_{\iota\to\alpha} g_\iota = h$, i.e. $S\colon g \overset{p}{\twoheadrightarrow}_\mathcal{R} h$. The "in $\mathcal{G}^\infty_\mathcal{C}(\Sigma)$" part follows from $S\colon g \overset{m}{\twoheadrightarrow}_\mathcal{R} h$.

We will also prove the "if" direction by induction on α: again, the case $\alpha = 0$ is trivial and the case that α is a successor ordinal follows immediately from the induction hypothesis.

Let α be a limit ordinal. As S is strongly p-convergent in $\mathcal{G}^\infty_\mathcal{C}(\Sigma)$, we know that $S|_\gamma\colon g \overset{p}{\twoheadrightarrow}_\mathcal{R} g_\gamma$ in $\mathcal{G}^\infty_\mathcal{C}(\Sigma)$ for all $\gamma < \alpha$. Thus, we can apply the induction hypothesis to obtain that $S|_\gamma\colon g \overset{m}{\twoheadrightarrow}_\mathcal{R} g_\gamma$ for each $\gamma < \alpha$. Hence, S is strongly m-continuous. As S strongly p-converges in $\mathcal{G}^\infty_\mathcal{C}(\Sigma)$, we know from Lemma 5.11 that $(\mathsf{depth}_{g_\iota}(n_\iota))_{\iota<\alpha}$ tends to infinity. With the strong m-continuity of S, this yields, according to Lemma 5.13, that S strongly m-converges to some h'. By Lemma 5.12, we conclude that $h' = h$, i.e. $S\colon g \overset{m}{\twoheadrightarrow}_\mathcal{R} h$. ◀

5.3 Normalisation of Strong p-convergence

In this section we shall show that – similarly to TRSs [4] – GRSs are normalising w.r.t. strong p-convergence. As for terms, this is a distinguishing feature of strong p-convergence. For example, the term graph rule (that unravels to) $c \to c$, for some constant c, yields a system in which c has no normal form w.r.t. strong m-convergence (or finite reduction or weak p-/m-convergence). If we consider strong p-convergence however, repeatedly applying the rule to c yields the normalising reduction $c \overset{p}{\twoheadrightarrow} \bot$. Term graphs which can be infinitely often contracted at the root – such as c – are called *root-active*:

▶ **Definition 5.15.** Let $\mathcal{R}$ be a GRS over Σ and $g \in \mathcal{G}^\infty_\mathcal{C}(\Sigma_\bot)$. Then g is called *root-active* if, for each reduction $g \overset{p}{\twoheadrightarrow}_\mathcal{R} g'$, there is a reduction $g' \overset{p}{\twoheadrightarrow}_\mathcal{R} h$ to a redex h in $\mathcal{R}$. The term graph g is called *root-stable* if, for each reduction $g \overset{p}{\twoheadrightarrow}_\mathcal{R} h$, h is not a redex in $\mathcal{R}$.

Similar to the construction of Böhm normal forms [18], the strategy for rewriting a term graph into normal form is to rewrite root-active sub-term graphs to $\bot$ and non-root-active sub-term graphs to root-stable terms. The following lemma will allow us to do that:

▶ **Lemma 5.16.** *Let $\mathcal{R}$ be a GRS over Σ and $g \in \mathcal{G}^\infty_\mathcal{C}(\Sigma_\bot)$.*

(i) If g is root-active, then there is a reduction $g \overset{p}{\twoheadrightarrow}_\mathcal{R} \bot$.

(ii) If g is not root-active, then there is a reduction $g \overset{p}{\twoheadrightarrow}_\mathcal{R} h$ to a root-stable term graph h.

(iii) If g is root-stable, then so is every term graph h with a reduction $g \overset{p}{\twoheadrightarrow}_\mathcal{R} h$.

In the following, we need to generalise the concatenation of sequences. To this end, we make use of the fact that the prefix order $\leq$ on sequences forms a cpo and thus has lubs for directed sets: let $(S_\iota)_{\iota<\alpha}$ be a sequence of sequences in a common set. The concatenation of $(S_\iota)_{\iota<\alpha}$, written $\prod_{\iota<\alpha} S_\iota$, is recursively defined as the empty sequence $\langle\rangle$ if $\alpha = 0$, $\left(\prod_{\iota<\alpha'} S_\iota\right) \cdot S_{\alpha'}$ if $\alpha = \alpha' + 1$, and $\bigsqcup_{\gamma<\alpha} \prod_{\iota<\gamma} S_\iota$ if α is a limit ordinal.

The following lemma shows that we can use the reductions from Lemma 5.16 in order to turn the sub-term graphs of a term graph into root-stable form level by level:

▶ **Lemma 5.17.** *Let $\mathcal{R}$ be a GRS over Σ, $g \in \mathcal{G}^\infty_\mathcal{C}(\Sigma_\bot)$ and $d < \omega$ such that $g|_n$ is root-stable for all $n \in N^g$ with $\mathsf{depth}_g(n) < d$. Then there is a reduction $S_d \colon g \overset{p}{\twoheadrightarrow}_\mathcal{R} h$ such that $h|_n$ is root-stable for each $n \in N^g$ with $\mathsf{depth}_g(n) \leq d$.*

Proof. There are only finitely many nodes in g at depth d, say, $n_0, n_1, \ldots, n_k$. Let π_i be a minimal position of n_i in g for each $i \leq k$. For each $i \leq k$, we construct a reduction $T_i \colon g_i \overset{p}{\twoheadrightarrow}_\mathcal{R} g_{i+1}$ with $g_0 = g$. Since all sub-term graphs at depth $< d$ are root stable, each step in T_i takes place at depth $\geq d$ and thus π_{i+1} is still a position in g_{i+1} of a node at depth d. If $g_i|_{\pi_i}$ is root-active, then Lemma 5.16 yields a reduction $g_i|_{\pi_i} \overset{p}{\twoheadrightarrow}_\mathcal{R} \bot$. Let T_i be the embedding of this reduction into g_i at position π_i. Hence, $g_{i+1}|_{\pi_i} = \bot$ is root-stable. If $g_i|_{\pi_i}$ is not root-active, then Lemma 5.16 yields a reduction $g_i|_{\pi_i} \overset{p}{\twoheadrightarrow}_\mathcal{R} g'_i$ to a root-stable term graph g'_i. Let T_i be the embedding of this reduction into g_i at position π_i. Hence, $g_{i+1}|_{\pi_i} = g'_i$ is root-stable.

Define $S_d \colon = \prod_{i\leq k} T_i$. Since, by Lemma 5.16, root-stability is preserved by strongly p-converging reductions, we can conclude that $S_d \colon g \overset{p}{\twoheadrightarrow}_\mathcal{R} g_{k+1}$ such that all sub-term graphs at depth at most d in g_{k+1} are root-stable. ◀

Note that the assumption that all sub-term graphs at depth $< d$ are root-stable is crucial. Otherwise, reductions within sub-term graphs at depth d may take place at depth $< d$!

Finally, the strategy for rewriting a term graph into normal form is to simply iterate the reductions that are given by Lemma 5.17 above.

▶ **Theorem 5.18.** *Every GRS $\mathcal{R}$ is normalising w.r.t. strongly p-converging reductions. That is, for each partial term graph g, there is a reduction $g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} h$ to a normal form h in $\mathcal{R}$.*

Proof. Given a partial term graph g_0, take the reductions $S_d\colon g_d \overset{p}{\twoheadrightarrow} g_{d+1}$ from Lemma 5.17 for each $d \in \mathbb{N}$ and construct $S = \prod_{d<\omega} S_d$. By Proposition 5.10, we have $S\colon g_0 \overset{p}{\twoheadrightarrow} g_\omega$ for some g_ω. As, by Lemma 5.16, root-stability is preserved by strongly p-converging reductions, and each reduction S_d increases the depth up to which sub-term graphs are root-stable, we know that each sub-term graph of g_ω is root-stable, i.e. g_ω is a normal form. ◀

The ability of strong p-convergence to normalise any term graph will be a crucial component of the proof of completeness of infinitary term graph rewriting.

6 Soundness and Completeness of Infinitary Term Graph Rewriting

In this section, we will study the relationship between GRSs and the corresponding TRSs they simulate. In particular, we will show the soundness of GRSs w.r.t. strong convergence and a restricted form of completeness. To this end we make use of the isomorphism between terms and canonical term trees as outlined at the end of Section 3.

▶ **Proposition 6.1.** *The unravelling $\mathcal{U}(g)$ of a term graph $g \in \mathcal{G}^\infty(\Sigma)$ is given by the labelled quotient tree $(\mathcal{P}(g), g(\cdot), \mathcal{I}_{\mathcal{P}(g)})$.*

Proof. Since $\mathcal{I}_{\mathcal{P}(g)}$ is a subrelation of $\sim_g$, we know that $(\mathcal{P}(g), g(\cdot), \mathcal{I}_{\mathcal{P}(g)})$ is a labelled quotient tree and thus uniquely determines a term t. By Lemma 3.7, there is a homomorphism from t to g. Hence, $\mathcal{U}(g) = t$. ◀

Before we start investigating the correspondences between term rewriting and term graph rewriting, we need to transfer the notions of left-linearity and orthogonality to GRSs:

▶ **Definition 6.2.** Let $\mathcal{R} = (\Sigma, R)$ be a GRS. A rule $\rho \in R$ is called *left-linear* resp. *left-finite* if its left-hand side ρ_l is a term tree resp. a finite term graph. The GRS $\mathcal{R}$ is called *left-linear* resp. *left-finite* if all its rules are left-linear resp. *left-finite*. The GRS $\mathcal{R}$ is called *orthogonal* if it is left-linear and the TRS $\mathcal{U}(\mathcal{R})$ is *non-overlapping*.

Note that the unravelling $\mathcal{U}(\mathcal{R})$ of a GRS $\mathcal{R}$ is left-linear if $\mathcal{R}$ is left-linear, that $\mathcal{U}(\mathcal{R})$ is left-finite if $\mathcal{R}$ is left-linear and left-finite, and that $\mathcal{U}(\mathcal{R})$ is orthogonal if $\mathcal{R}$ is orthogonal.

We have to single out a particular kind of redex that manifests a peculiar behaviour:

▶ **Definition 6.3.** A redex of a rule (g, l, r) is called *circular* if l and r are distinct but the matching $\mathcal{V}$-homomorphism ϕ maps them to the same node, i.e. $l \neq r$ but $\phi(l) = \phi(r)$.

Kennaway et al. [16] show that circular redexes only reduce to themselves:

▶ **Proposition 6.4.** *For every circular ρ-redex $g|_n$, we have $g \mapsto_{n,\rho} g$.*

However, contracting the unravelling of a circular redex also yields the same term:

▶ **Lemma 6.5.** *For every circular ρ-redex $g|_n$, we have $\mathcal{U}(g) \to_{\pi,\mathcal{U}(\rho)} \mathcal{U}(g)$ for all $\pi \in \mathcal{P}_g(n)$.*

Proof. Since there is a circular ρ-redex, we know that the right-hand side root r^ρ is reachable but different from the left-hand side root l^ρ of ρ. Hence, there is a non-empty path $\widehat{\pi}$ from l^ρ to r^ρ. Because $g|_n$ is a circular ρ-redex, the corresponding matching $\mathcal{V}$-homomorphism maps both l^ρ and r^ρ to n. Since Δ-homomorphisms preserve paths, we thus know that $\widehat{\pi}$ is also a path from n to itself in g. In other words, $\pi \in \mathcal{P}_g(n)$ implies $\pi \cdot \widehat{\pi} \in \mathcal{P}_g(n)$. Consequently, for each $\pi \in \mathcal{P}_g(n)$, we have that $\mathcal{U}(g)|_\pi = \mathcal{U}(g)|_{\pi\cdot\widehat{\pi}}$.

Since there is a path $\widehat{\pi}$ from l^ρ to r^ρ, the unravelling $\mathcal{U}(\rho)$ of ρ is of the form $s \to s|_{\widehat{\pi}}$. Hence, we know that each application of $\mathcal{U}(\rho)$ at a position π in some term t replaces the subterm at π with the subterm at $\pi \cdot \widehat{\pi}$ in t, i.e. $t \to_{\pi,\mathcal{U}(\rho)} t[t|_{\pi\cdot\widehat{\pi}}]_\pi$.

Combining the two findings above, we obtain that

$$\mathcal{U}(g) \to_{\pi,\mathcal{U}(\rho)} \mathcal{U}(g)[\mathcal{U}(g)|_{\pi\cdot\widehat{\pi}}]_\pi = \mathcal{U}(g)[\mathcal{U}(g)|_\pi]_\pi = \mathcal{U}(g) \quad \text{for all } \pi \in \mathcal{P}_g(n) \qquad ◀$$

The following two properties due to Kennaway et al. [16] show how single term graph reduction steps relate to term reductions in the corresponding unravelling.[1]

▶ **Proposition 6.6.** *Given a left-linear GRS $\mathcal{R}$ and a term graph g in $\mathcal{R}$, it holds that g is a normal form in $\mathcal{R}$ iff $\mathcal{U}(g)$ is a normal form in $\mathcal{U}(\mathcal{R})$.*

▶ **Theorem 6.7.** *Let $\mathcal{R}$ be a left-linear, left-finite GRS with a reduction step $g \to_{n,\rho} h$. Then $S\colon \mathcal{U}(g) \overset{m}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$ such that the depth of every redex contracted in S is greater or equal to $\mathsf{depth}_g(n)$. In particular, if the ρ-redex $g|_n$ is not circular, then S is a complete development of the set of redex occurrences $\mathcal{P}_g(n)$ in $\mathcal{U}(g)$.*

In the following, we will generalise the above soundness theorem to strongly p-converging term graph reductions. We will then use the correspondence between strong m-convergence and strong p-convergence in $\mathcal{G}^\infty_\mathcal{C}(\Sigma)$ to transfer that result to strongly m-converging reductions.

At first, we can observe that the limit inferior commutes with the unravelling:

▶ **Proposition 6.8.** *For each sequence $(g_\iota)_{\iota<\alpha}$ in the partially ordered set $(\mathcal{G}^\infty_\mathcal{C}(\Sigma_\perp), \leq^\mathsf{S}_\perp)$, we have that $\mathcal{U}(\liminf_{\iota\to\alpha} g_\iota) = \liminf_{\iota\to\alpha} \mathcal{U}(g_\iota)$.*

Proof. This is an immediate consequence of Theorem 4.2 and Proposition 6.1. ◀

In order to prove soundness w.r.t. strong p-convergence, we need to turn the statement about the depth of redexes in Theorem 6.7 into a statement about the corresponding reduction contexts. To this end, we make use of the fact that the semilattice structure of $\leq^\mathsf{S}_\perp$ admits greatest lower bounds for non-empty sets of term graphs:

▶ **Proposition 6.9** ([6]). *In the partially ordered set $(\mathcal{G}^\infty_\mathcal{C}(\Sigma_\perp), \leq^\mathsf{S}_\perp)$ every non-empty set G has a greatest lower bound $\bigsqcap G$ given by the following labelled quotient tree $(P, l, \sim)$:*

$$P = \left\{\pi \in \bigcap\nolimits_{g\in G} \mathcal{P}(g) \,\middle|\, \forall \pi' < \pi \exists f \in \Sigma_\perp \forall g \in G : g(\pi') = f \right\}$$

$$l(\pi) = \begin{cases} f & \text{if } \forall g \in G : f = g(\pi) \\ \perp & \text{otherwise} \end{cases} \qquad\qquad \sim \; = \bigcap\nolimits_{g\in G} \sim_g \cap\, P \times P$$

In particular, the glb of a set of term trees is again a term tree.

We can then prove the following proposition that relates the reduction context of a term graph reduction step with the reduction contexts of the corresponding term reduction:

▶ **Proposition 6.10.** *For each reduction step $g \to_c h$ in a left-linear, left-finite GRS $\mathcal{R}$, there is a non-empty reduction $S = (t_\iota \to_{c_\iota} t_{\iota+1})_{\iota<\alpha}$ with $S\colon \mathcal{U}(g) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$ and $\mathcal{U}(c) = \bigsqcap_{\iota<\alpha} c_\iota$.*

[1] The original results are on finite term graphs. However, for the correspondence of normal forms, this restriction is not necessary, and for the soundness, only the finiteness of left-hand sides is crucial.

Proof. By Theorem 6.7, there is a reduction $S\colon \mathcal{U}(g) \overset{m}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$. At first we assume that the redex $g|_n$ contracted in $g \to_n h$ is not a circular redex. Hence, S is a complete development of the set of redex occurrences $\mathcal{P}_g(n)$ in $\mathcal{U}(g)$. By Theorem 2.3, we then obtain $S\colon \mathcal{U}(g) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$. From Lemma 5.5 and Proposition 6.1 it follows that $\mathcal{U}(g\backslash n)$ is obtained from $\mathcal{U}(g)$ by replacing each subterm of $\mathcal{U}(g)$ at a position in $\mathcal{P}^m_g(n)$, i.e. a minimal position of n, by $\bot$. Since each step $t_\iota \to_{\pi_\iota} t_{\iota+1}$ in S contracts a redex at a position π_ι that has a prefix in $\mathcal{P}^m_g(n)$, we have, by Proposition 6.9 and Corollary 4.1, that $\mathcal{U}(g\backslash n) \leq^{\mathsf{S}}_\bot \bigsqcap_{\iota<\alpha} t_\iota[\bot]_{\pi_\iota} = \bigsqcap_{\iota<\alpha} c_\iota$. Moreover, for each $\pi \in \mathcal{P}^m_g(n)$ there is a step at $\iota_\pi < \alpha$ in S that takes place at π. From Proposition 6.9, it is thus clear that $\mathcal{U}(g\backslash n) = \bigsqcap_{\pi\in\mathcal{P}^m_g(n)} c_{\iota_\pi}$, which means that $\mathcal{U}(g\backslash n) \geq^{\mathsf{S}}_\bot \bigsqcap_{\iota<\alpha} c_\iota$. Due to the antisymmetry of $\leq^{\mathsf{S}}_\bot$, we thus know that $\mathcal{U}(g\backslash n) = \bigsqcap_{\iota<\alpha} c_\iota$. Then $\mathcal{U}(c) = \bigsqcap_{\iota<\alpha} c_\iota$ follows from the fact that $c \cong g\backslash n$.

If the ρ-redex $g|_n$ contracted in $g \to_{\rho,n} h$ is a circular redex, then $g = h$ according to Proposition 6.4. However, by Lemma 6.5, each $\mathcal{U}(\rho)$-redex at positions in $\mathcal{P}_g(n)$ in $\mathcal{U}(g)$ reduces to itself as well. Hence, we get a reduction $\mathcal{U}(g) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\rho)} \mathcal{U}(h)$ via a complete development of the redexes at the minimal positions $\mathcal{P}^m_g(n)$ of n in g. The equality $\mathcal{U}(c) = \bigsqcap_{\iota<\alpha} c_\iota$ then follows as for the first case above. ◀

In order to prove the soundness of strongly p-converging term graph reductions, we need the following technical lemma, which can be proved easily:

▶ **Lemma 6.11.** *Let $(a_\iota)_{\iota<\alpha}$ be a sequence in a complete semilattice $(A, \leq)$ and $(\gamma_\iota)_{\iota<\delta}$ a strictly monotone sequence in the ordinal α such that $\bigsqcup_{\iota<\delta} \gamma_\iota = \alpha$. Then*

$$\liminf_{\iota\to\alpha} a_\iota = \liminf_{\beta\to\delta} \left(\bigsqcap_{\gamma_\beta\leq\iota<\gamma_{\beta+1}} a_\iota\right).$$

▶ **Theorem 6.12.** *Let $\mathcal{R}$ be a left-linear, left-finite GRS. If $g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} h$, then $\mathcal{U}(g) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$.*

Proof. Let $S = (g_\iota \to_{c_\iota} g_{\iota+1})_{\iota<\alpha}$ be a reduction strongly p-converging to g_α in $\mathcal{R}$. By Proposition 6.10, there is, for each $\gamma < \alpha$, a reduction $T_\gamma\colon \mathcal{U}(g_\gamma) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(g_{\gamma+1})$ such that

$$\bigsqcap_{\iota<|T_\gamma|} \bar{c}_\iota = \mathcal{U}(c_\gamma), \quad \text{where } (\bar{c}_\iota)_{\iota<|T_\gamma|} \text{ is the sequence of reduction contexts in } T_\gamma. \qquad (*)$$

Define for each $\delta \leq \alpha$ the concatenation $U_\delta = \prod_{\iota<\delta} T_\iota$. We will show that $U_\delta\colon \mathcal{U}(g_0) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(g_\delta)$ for each $\delta \leq \alpha$ by induction on δ. The theorem is then obtained from the case $\delta = \alpha$.

The case $\delta = 0$ is trivial, and the case $\delta = \delta' + 1$ follows from the induction hypothesis.

For the case that δ is a limit ordinal, let $U_\delta = (t_\iota \to_{c'_\iota} t_{\iota+1})_{\iota<\beta}$. For each $\gamma < \beta$ we find some $\delta' < \delta$ with $U_\delta|_\gamma < U_{\delta'}$. By induction hypothesis, we can assume that $U_{\delta'}$ is strongly p-continuous. Thus, the proper prefix $U_\delta|_\gamma$ strongly p-converges to t_γ. This shows that each proper prefix $U_\delta|_\gamma$ of U_δ strongly p-converges to t_γ. Hence, U_δ is strongly p-continuous.

In order to show that $U_\delta\colon \mathcal{U}(g_0) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(g_\delta)$, it remains to be shown that $\liminf_{\iota\to\beta} c'_\iota = \mathcal{U}(g_\delta)$. Since S is strongly p-converging, we know that $\liminf_{\iota\to\delta} c_\iota = g_\delta$. By Proposition 6.8, we thus have $\liminf_{\iota\to\delta} \mathcal{U}(c_\iota) = \mathcal{U}(g_\delta)$. By $(*)$ and the construction of U_δ, there is a strictly monotone sequence $(\gamma_\iota)_{\iota<\delta}$ with $\gamma_0 = 0$ and $\bigsqcup_{\iota<\delta} \gamma_\iota = \beta$ such that $\mathcal{U}(c_\iota) = \bigsqcap_{\gamma_\iota\leq\gamma<\gamma_{\iota+1}} c'_\gamma$ for all $\iota < \delta$. Thus, we can complete the proof as follows:

$$\mathcal{U}(g_\delta) = \liminf_{\iota\to\delta} \mathcal{U}(c_\iota) = \liminf_{\iota\to\delta} \left(\bigsqcap_{\gamma_\iota\leq\gamma<\gamma_{\iota+1}} c'_\gamma\right) \overset{\text{Lem. 6.11}}{=} \liminf_{\iota\to\beta} c'_\iota$$ ◀

By combining the soundness result above with the normalisation of strong p-convergence, we obtain the following completeness result:

▶ **Theorem 6.13.** *Given an orthogonal, left-finite GRS $\mathcal{R}$, we obtain for each reduction $\mathcal{U}(g) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} t$, a reduction $g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} h$ such that $t \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$.*

Proof. Let $\mathcal{U}(g) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} t$. By Theorem 5.18 there is a normalising reduction $g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} h$. According to Theorem 6.12, $g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} h$ implies $\mathcal{U}(g) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$. By Proposition 6.6, $\mathcal{U}(h)$ is a normal form in $\mathcal{U}(\mathcal{R})$. Since orthogonal, left-finite TRSs are confluent w.r.t. strong p-convergence [4], the reduction $\mathcal{U}(g) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$ together with $\mathcal{U}(g) \overset{m}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} t$ yields a reduction $t \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$. ◀

The results above make strongly p-converging term graph reductions sound and complete for strongly p-converging term reductions in the sense of *adequacy* of Kennaway et al. [16].

The notion of adequacy of Kennaway et al. [16] does not only comprise soundness and completeness but also demands that the unravelling $\mathcal{U}(\cdot)$ is surjective and both preserves and reflects normal forms. For infinitary term graph rewriting, surjectivity of $\mathcal{U}(\cdot)$ is trivial since each term is the image of itself under $\mathcal{U}(\cdot)$ and the preservation and reflection of normal forms is given for left-linear GRSs by Proposition 6.6.

From the soundness result for strong p-convergence, we can straightforwardly derive a corresponding result for strong m-convergence:

▶ **Theorem 6.14.** *Let $\mathcal{R}$ be a left-linear, left-finite GRS. If $g \overset{m}{\twoheadrightarrow}_{\mathcal{R}} h$, then $\mathcal{U}(g) \overset{m}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$.*

Proof. Given a reduction $S\colon g \overset{m}{\twoheadrightarrow}_{\mathcal{R}} h$, we know, by Theorem 5.14, that $S\colon g \overset{p}{\twoheadrightarrow}_{\mathcal{R}} h$ in $\mathcal{G}^{\infty}_{\mathcal{C}}(\Sigma)$. According to Theorem 6.12, we then find a reduction $\mathcal{U}(g) \overset{p}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$. Since, g, h are total, so are $\mathcal{U}(g), \mathcal{U}(h)$. Hence, by Corollary 7.15 of [4], we obtain a reduction $\mathcal{U}(g) \overset{m}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$. ◀

Similar to the proof of Theorem 6.13, we can derive a weakened completeness property for strong m-convergence:

▶ **Theorem 6.15.** *Given an orthogonal, left-finite GRS $\mathcal{R}$ that is normalising w.r.t. strongly m-converging reductions, we find for each normalising reduction $\mathcal{U}(g) \overset{m}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} t$ a reduction $g \overset{m}{\twoheadrightarrow}_{\mathcal{R}} h$ such that $t = \mathcal{U}(h)$.*

Proof. Let $\mathcal{U}(g) \overset{m}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} t$ with t a normal form in $\mathcal{U}(\mathcal{R})$. As $\mathcal{R}$ is normalising w.r.t. strongly m-converging reductions, there is a reduction $g \overset{m}{\twoheadrightarrow}_{\mathcal{R}} h$ with h a normal form in $\mathcal{R}$. According to Theorem 6.14, we then find a reduction $\mathcal{U}(g) \overset{m}{\twoheadrightarrow}_{\mathcal{U}(\mathcal{R})} \mathcal{U}(h)$. By Proposition 6.6, $\mathcal{U}(h)$ is a normal form in $\mathcal{U}(\mathcal{R})$. Since $\mathcal{U}(\mathcal{R})$ is left-finite and orthogonal, we know that, according to Theorem 7.15 in [17], $\mathcal{R}$ has unique normal forms w.r.t. $\overset{m}{\twoheadrightarrow}$. Consequently, $t = \mathcal{U}(h)$. ◀

While the above theorem is restricted to normalising GRSs, we conjecture that this restriction is not needed: as soon as we have a compression lemma for strong p-convergence, completeness of normalising strong m-convergence follows from the completeness of strong p-convergence.

Yet, as mentioned in the in the introduction, the restriction to normalising reductions is crucial. The counterexample that Kennaway et al. [16] give for their informal notion of term graph convergence in fact also applies to our notion of strong m-convergence.

7 Conclusions

By generalising the metric and partial order based notions of convergence from terms to term graphs, we have obtained two infinitary term graph rewriting calculi that simulate infinitary term rewriting adequately. Not only do these results show the appropriateness of our notions of infinitary term graph rewriting. They also refute the claim of Kennaway et al. [16] that infinitary term graph rewriting cannot adequately simulate infinitary term rewriting.

Since reasoning over the rather operational style of term graph rewriting is tedious, we tried to simplify the proofs using labelled quotient trees. In future work, it would be helpful to characterise term graph rewriting itself in this way or to adopt a more declarative approach to term graph rewriting [12, 11, 1].

We think that, in this context, strong p-convergence may help to bridge the differences between the operational style of Barendregt et al. [8] and the declarative formalisms [12, 11, 1], which arise from the different way of contracting circular redexes. While in the operational approach that we adopted here, circular redexes are contracted to themselves, they are contracted to $\perp$ in the abovementioned declarative approaches. However, since circular redexes are root-active, they can be rewritten to $\perp$ in a strongly p-converging reduction.

References

1 Z.M. Ariola and J.W. Klop. Equational term graph rewriting. *Fund. inform.*, 26(3-4):207–240, 1996.

2 A. Arnold and M. Nivat. The metric space of infinite trees. Algebraic and topological properties. *Fund. inform.*, 3(4):445–476, 1980.

3 P. Bahr. Abstract models of transfinite reductions. In C. Lynch, editor, *RTA 2010*, volume 6 of *LIPIcs*, pages 49–66. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010.

4 P. Bahr. Partial order infinitary term rewriting and Böhm trees. In C. Lynch, editor, *RTA 2010*, volume 6 of *LIPIcs*, pages 67–84. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010.

5 P. Bahr. Modes of convergence for term graph rewriting. In Manfred Schmidt-Schauß, editor, *RTA 2011*, volume 10 of *LIPIcs*, pages 139–154, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

6 P. Bahr. Convergence in infinitary term graph rewriting systems is simple. Unpublished manuscript, available from the author's web site, 2012.

7 P. Bahr. Infinitary term graph rewriting is simple, sound and complete. Companion report, available from the author's web site, 2012.

8 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 P.C. Treleaven J. de Bakker, A. J. Nijman, editor, *PARLE 1987*, volume 259 of *LNCS*, pages 141–158. Springer, 1987.

9 E. Barendsen. Term graph rewriting. In Terese [19], chapter 13, pages 712–743.

10 J. Bongaerts. *Topological Convergence in Infinitary Abstract Rewriting*. Master's Thesis, Utrecht University, 2011.

11 A. Corradini and F. Drewes. Term graph rewriting and parallel term rewriting. In *TERMGRAPH '11*, pages 3–18, 2011.

12 A. Corradini and F. Gadducci. Rewriting on cyclic structures: Equivalence between the operational and the categorical description. *RAIRO Theor. Inf. Appl.*, 33(4):467–493, 1999.

13 J.A. Goguen, J.W. Thatcher, E.G. Wagner, and J.B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24(1):68–95, 1977.

14 J.L. Kelley. *General Topology*, volume 27 of *Graduate Texts in Mathematics*. Springer-Verlag, 1955.

15 R. Kennaway and F.-J. de Vries. Infinitary rewriting. In Terese [19], chapter 12, pages 668–711.

16 R. Kennaway, J.W. Klop, M.R. Sleep, and F.-J. de Vries. On the adequacy of graph rewriting for simulating term rewriting. *ACM Prog. Lang. Sys.*, 16(3):493–523, 1994.

17 R. Kennaway, J.W. Klop, M.R. Sleep, and F.-J. de Vries. Transfinite reductions in orthogonal term rewriting systems. *Ann. Pure Appl. Logic*, 119(1):18–38, 1995.

18 R. Kennaway, V. van Oostrom, and F.-J. de Vries. Meaningless terms in rewriting. *J. Funct. Logic Programming*, 1999(1):1–35, February 1999.

19 Terese. *Term Rewriting Systems*. Cambridge University Press, 1st edition, 2003.

Axiomatic Sharing-via-Labelling

Thibaut Balabonski

Univ Paris Diderot, Sorbonne Paris Cité,
PPS, UMR 7126, CNRS,
F-75205 Paris, France
`thibaut.balabonski@pps.univ-paris-diderot.fr`

Abstract

A judicious use of labelled terms makes it possible to bring together the simplicity of term rewriting and the sharing power of graph rewriting: this has been known for twenty years in the particular case of orthogonal first-order systems. The present paper introduces a concise and easily usable axiomatic presentation of sharing-via-labelling techniques that applies to higher-order term rewriting as well as to non-orthogonal term rewriting. This provides a general framework for the sharing of subterms and keeps the formalism as simple as term rewriting.

1998 ACM Subject Classification I.1.3 Languages and Systems: Evaluation strategies

Keywords and phrases Sharing, Abstract term rewriting, Graphs, Higher order, Non-orthogonality

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.85

Category Regular Research Paper

1 Introduction

Termgraph rewriting [9, 27] improves on term rewriting by allowing the sharing of subterms, thus preventing some duplications of computation steps. However, graph formalisms are usually far more complex than the corresponding term formalisms. Higher-order graphs [29, 11, 19] in particular require non-trivial correctness criteria and readback procedures.

This paper aims at providing a fully general framework for the sharing of subterms, while keeping the formalism as simple as term rewriting. It substantially generalizes an elegant partial solution originating in L. Maranget's work [25], of which we name two features:

- Graphs are represented by labelled terms.
- Graph reduction is simulated by sequences of term reduction.

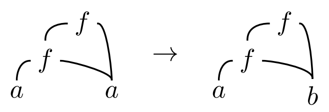
Thus we get a formal setting for graph rewriting using only the well-known term rewriting.

This started 20 years ago with orthogonal first-order rewriting [25], and has then been extended to several weak λ-calculi [10, 7] or orthogonal extensions of these weak calculi [6]. Thanks to its technological simplicity the approach turned out to be useful as an analysis tool for various purposes, in particular by allowing the description of optimal reduction strategies [25] and by allowing simple comparisons between different graph implementations of the λ-calculus [7].

In this line of work, any symbol in a term is given a label which is a metaphorical location. Two subterms with the same label are supposed to be physically equal, stored at the same location in memory or drawn at the same coordinate in a picture (see Example 1). Graph rewriting is then simulated by the *simultaneous* reduction of all the term-redexes corresponding to a unique graph-redex, which means all the term-redexes with a given label.

▶ **Example 1.** The two occurrences of a^δ denote the same node and are reduced simultaneously. Across this paper we use a single arrow $\to$ for the reduction of one redex, and a double arrow $\Rightarrow$ for the simultaneous reduction of several redexes.

$$\begin{array}{rl} & f^\alpha(f^\beta(a^\gamma, a^\delta), a^\delta) \\ \Rightarrow & f^\alpha(f^\beta(a^\gamma, b^\zeta), b^\zeta) \end{array}$$



◀

This correspondence is sound as long as the system preserves a *sharing property* on its labelled terms: whenever two subterms coexist with the same label they are syntactically equal. Note that in this setting, a labelled term t satisfying the sharing property is itself a witness of the correctness of the corresponding graph $\mathcal{G}_t$, and that $\mathcal{G}_t$ is an *acyclic graph.*

While simultaneous reduction in general is a non-trivial extension to one-step rewriting, the particular case considered here is simply definable within usual term rewriting. Indeed we consider the simultaneous reduction of *disjoint* subterms, which we call *parallel reduction.* It is simulated by the sequential reduction of the subterms in any order.

▶ **Example 2.** The parallel reduction of Example 1 can be implemented by:
$f^\alpha(f^\beta(a^\gamma, a^\delta), a^\delta) \quad \to \quad f^\alpha(f^\beta(a^\gamma, b^\zeta), a^\delta) \quad \to \quad f^\alpha(f^\beta(a^\gamma, b^\zeta), b^\zeta)\,.$ ◀

In [25, 10, 6, 7] the labels are generated in an effective way, which helps to define the labelled reduction by simple term rewriting rule schemes. The generation of labels in a reduction is in two steps: first the redex is given a name Ω based on its labels, then new labels are generated using Ω. For instance in Example 2, $\Omega = \delta$ and $\zeta = \,<\Omega, \zeta_0>$ for some ζ_0 fixed in the rule scheme.

In this method inspired by Lévy's labelling for optimal sharing [23], the name of a redex reflects the past reductions *causally* related to this redex, and is *invariant* from the creation of the redex to its disappearance. These are key ingredients for providing absolute bounds on the sharing of redexes, which is the point of Lévy's optimality theory. In [25, 10, 6, 7], both of these points also seem to play an important role in the preservation of the sharing property. This limits the applicability of these labellings for sharing in several ways:

Orthogonality For the redex names to remain static, no reduction should ever interfere with an existing redex in a way that could alter its name. Hence the restriction to orthogonal systems where no two reduction rules can apply at overlapping sets of positions.

Causal sharing For each system the causal constraints of the previous labellings essentially determine one level of sharing. Hence defining other sharing disciplines requires tampering with the dynamics of the system. It is done in [7] by introducing various fine-tuned notions of weak reduction (systems where reduction in the scope of a binder is constrained).

Weak reduction In higher-order rewriting, the "causal sharing" is typically incompatible with our acyclic graphs, as the optimal sharing of the λ-calculus. Thus the previous approaches apply to higher-order only through a restriction to weak systems that yield compatible sharings. While the restriction does not prevent one from expressing the most common evaluation strategies of the λ-calculus, it seems to rule out any truly higher-order behaviour. Indeed, two common weak λ-calculi have been shown in [21, 7] to be strongly equivalent to orthogonal first-order rewriting systems.

This paper generalizes the previous approaches, in particular by freeing them of their bond to Lévy's optimality theory. It expresses sharing-via-labelling in an abstract rewriting framework encompassing any term rewriting system, be it higher-order and non-orthogonal. In this framework we show that the preservation of sharing needs neither a static identification of redexes, nor a tight relation to causality. We thus break through the previous limitations by giving to redexes a *dynamic* identity. And this makes all the difference!

Our main contribution is an axiomatic framework for the sharing of subterms that is:

Simple: it does not require more formal technology than term rewriting.
Expressive: it can describe a variety of graph algorithms for any term-style rewriting system.
Usable: it relies only on few axioms, and comes with highly adaptable concrete examples.
As a second contribution, we use the axiomatic framework to revisit call-by-need reduction and improve its sharing (Section 5).

Section 2 gives an overview of our axiomatic framework and its main mechanisms, and discusses its expressive power. Section 3 provides all the formal definitions and proofs. Section 4 presents some simple examples, and Section 5 develops a more elaborate example that takes advantage of the specifities of our approach. Other approaches to sharing and to graph reduction are reviewed in Section 6, then Section 7 concludes.

2 Overview of the axiomatic framework

This section gives an informal presentation of our axiomatic framework and discusses its mechanisms. Section 2.1 sets the scene of an abstract term rewriting framework. Section 2.2 presents the mechanisms of our axiomatic labelling on an example, and Section 2.3 discusses the expressive power of the whole framework.

2.1 Principles of the basic formalism

Abstract Rewriting Systems (ARS) [26, 28] give a fully general description of rewriting. In [26] any system is described by only four basic elements: a set $\mathcal{O}$ of objects, a set $\mathcal{R}$ of reductions, and two functions $\texttt{src} : \mathcal{R} \to \mathcal{O}$ and $\texttt{tgt} : \mathcal{R} \to \mathcal{O}$ assigning a source object and a target object to each reduction. Hence *ARS* can express rewriting on any kind of objects through any kind of rewriting rules with any kind of rule application conditions.

We introduce *Abstract Term Rewriting Systems (ATRS*, Definition 9*)* as a specialization of *ARS* to term rewriting by taking as set of objects $\mathcal{O}$ a set of terms. Now each reduction r is supposed to apply to a particular subterm of the source, at a position $\texttt{root}(r)$ called root of r, and to leave the rest unchanged (see Fig. 1).

Equivalently, any reduction r can be identified by three other components (see Fig. 2): the redex $\texttt{dex}(r)$ and reduct $\texttt{duct}(r)$ are the source-version and target-version of the modified subterm and the context $\texttt{ctx}(r)$ is what remains and is common to the source and the target.

2.2 Principles of the axiomatic labelling

In this section we derive from scratch a labelling for a higher-order rewriting system. By showing the backstage of this sample construction we present the main ideas of the axiomatic labelling defined in Section 3 (Definition 17).

We consider the fixpoint operator expressed by the rule scheme: $\mu x.t \;\to\; t^{\{x := \mu x.t\}}$. We analyse the following graph reduction, where the symbol h is duplicated but the expression $g(a)$ is kept shared.

$$\begin{array}{rll} & f(g(a),\ \underline{\mu x.h(g(a),x)}\ ) & (t_1) \\ \to & f(g(a),\ \underline{h(g(a),\mu x.h(g(a),x))}\ ) & (t_2) \end{array}$$

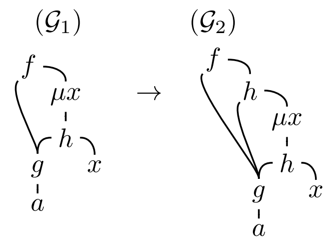


Figure 1 Abstract term reduction, ARS-style: source, target, root.

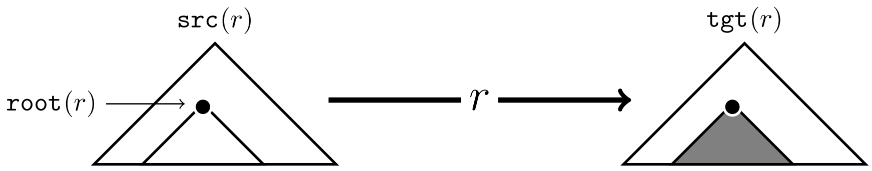
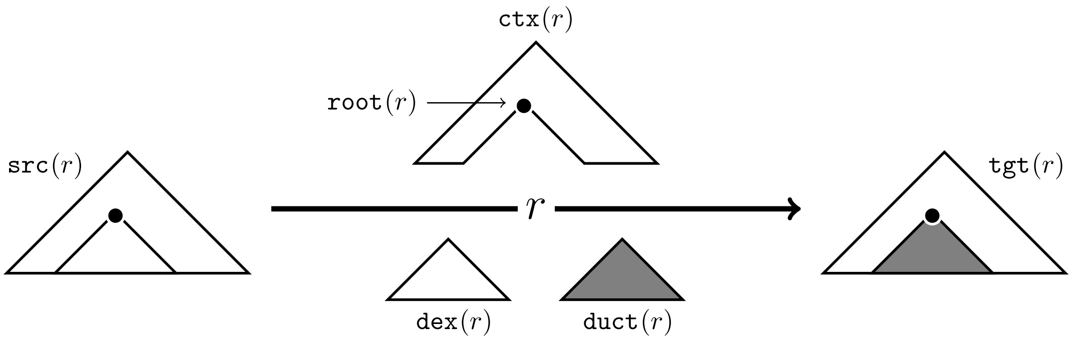
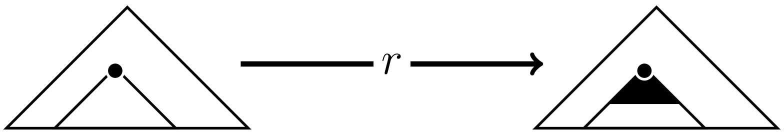


Figure 2 Abstract term reduction, TRS-style: redex, reduct, context.



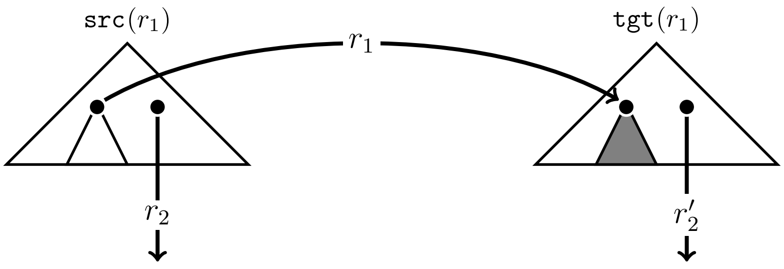
Remark that either triple $(\texttt{src}, \texttt{tgt}, \texttt{root})$ or $(\texttt{dex}, \texttt{duct}, \texttt{ctx})$ is enough to deduce the other informations. For instance, decomposing the source at the root position gives the context and the redex. Conversely, combining the context and the reduct yields the target.

Figure 3 Effect zone



The effect zone, in black, is a connected part of the reduct that contains all that is created or modified by the reduction.

Figure 4 Reductions at disjoint positions



The reduction r_1 cannot suppress the disjoint reduction r_2: some similar r_2' remains in the target of r_1. An equivalence of reductions (Definition 9) relates r_2 and r_2'.

We want to represent the graph $\mathcal{G}_1$ by a labelling t_1^l of the term t_1 such that each label denotes a node of the graph. We label the symbols f, g and h. In particular the labels of the two occurrences of g have to be equal, and the other labels have to be different. For instance:

$$f^{\alpha}(g^{\beta}(a),\ \mu x.h^{\gamma}(g^{\beta}(a),x)\ ) \qquad (t_1^l)$$

Now the reduction of t_1^l will act on the labels along the following principles. If labels figuratively represent memory locations, then label generation corresponds to memory allocation. Hence the new labels correspond to creations or to duplications of graph nodes. Call effect zone and write $\texttt{effz}(r)$ the part of the target of a reduction r where new labels are created (see Fig. 3). Intuitively, a smaller effect zone tends to make less duplications.

First, we do not want the reduction to modify its context: the effect zone is contained in the reduct $h(g(a), \mu x.h(g(a), x))$ and the labelled context $f^{\alpha}(g^{\beta}(a), \_)$ keeps its labels.

Second, remark that some subterms of the reduct are exact copies of some subterms of the redex. This is the case for instance of $h(g(a), x)$, or of the first occurrence of $g(a)$. We do not need to make new copies of these subterms. Thus they can be kept outside of the effect zone and inherit the labels of their source-side counterparts. Axiom *Effect zone* in Definition 13 checks that any other position is in the effect zone.

Finally, only the outer occurrence of h needs a new label, say δ. Then we can take the following labelling t_2^l of t_2 to represent $\mathcal{G}_2$:

$$f^{\alpha}(g^{\beta}(a),\ h^{\delta}(g^{\beta}(a), \mu x.h^{\gamma}(g^{\beta}(a),x))\ ) \qquad (t_2^l)$$

At this point, we have to generate a new label δ in a way that does not break the sharing property. Preferably, we would like a method that deduces the new labels from the redex and does not require any global information on the context. This fundamental choice of design allows us to define labelled reduction as a usual term rewriting rule scheme.

To avoid unintended clashes, the idea is to blend into the new label δ some information that is characteristic of the reduced redex. As a consequence, two distinct redexes will generate different labels, while two copies of the same redex will naturally have similar evolutions. In our metaphor a safe such characterization of a redex is its physical identity, in other words its label. This choice is possible if and only if the root of the reduced redex is labelled, which is required by an axiom *Redex head label*. In our example, we add a label ω to the symbol μx in t_1^l and get the following corrected labelling of t_1:

$$f^{\alpha}(g^{\beta}(a),\ \mu x^{\omega}.h^{\gamma}(g^{\beta}(a),x)\ ) \qquad (t_1^l)$$

Then the label δ can be explicitly built from ω and other information taken in the redex: say for instance that h^{δ} is the copy of h^{γ} ordered by a redex of label ω, and write $\delta = [\omega]\gamma$.

Such a generation method is sound if the following last requirement is met: once a redex of label ω is reduced, creating compounds such as $[\omega]\gamma$, no other redex labelled by ω should ever be created. Or better, no new occurrences of the label ω should ever be generated.

A first, short-term, precaution is to change the label of μx, which is done by putting μx in the effect zone. This is one requirement of the axiom *Effect zone*: the copies of the root of the redex must be in the effect zone. Also remark that the new label of μx, for instance $[\omega]$, has to be taken different from the new label δ created for h. An axiom *Separation* states that the labels created in the same reduction do not clash.

Then we need to check this property on the long run. We use two new ingredients:

- A strict order $\hookrightarrow$ on labels called *contribution* such that $\alpha \hookrightarrow [\alpha]$ and $\alpha \hookrightarrow [\alpha]\beta$ for any two labels α and β. An axiom *Contribution* expresses that the generated labels progress along the contribution order.

- An *independence* invariant on labelled terms that forbids the existence in a term of two labels α and β such that $\alpha \hookrightarrow \beta$ (*Independence property*, Definition 19).

In this setting we have all the ingredients needed to prove that the sharing property is preserved by parallel reduction (Sharing theorem 21).

2.3 Expressive power

Our framework axiomatizes sharing-via-labelling along three directions: the underlying term systems, the graph algorithms, and the labellings. Each of these levels of abstraction brings its own expressive power. They are reviewed here in reverse order.

The labels play two roles in our framework. As announced they first describe sharing, but they moreover have to contain enough information to allow the generation of fresh labels. The axiom *Contribution* we use for this is strictly less restrictive than its equivalent in the previous approaches. In particular our labels need not contain as much causal information as their predecessors, and our framework covers the previous labellings as well as new and possibly simpler ones that are illustrated across this paper (see Section 4.2). In particular, the sequences of labels and the dummy nodes that create arbitrary long chains of indirections in [10, 7] are no longer necessary. This is possible thanks to an invariant of *independence* that is strictly more general than its predecessors, and that is a key of our technical contribution.

The graph algorithms that can be described in our framework come from the combination of two parameters: first the effect zone specifies the positions for which new nodes are created, then the labelling decides on the positions of the effect zone that are to be shared or not. The first parameter is constrained by axiom *Effect zone* and the second by axiom *Separation*, which are both expressed in terms of equality of subterms. This allows us to describe a variety of graph reduction systems, such as the many variants of call-by-need, lazy, and fully lazy reduction that have been studied on the λ-calculus or similar systems. Some have been reviewed in [7] and encoded in a sharing-via-labelling framework, at the cost of defining finely-tuned reduction strategies. Such studies can be carried on in our new framework without tampering with the dynamics of the systems (see Section 4.2).

Our abstract term rewriting framework makes no assumption on the shape of the reduction rules of the underlying term systems, and few on the application conditions of these rules. The only restriction is the fairly natural one that the reductions affecting disjoint subterms of the same source term do not interfere (see Fig. 4). This allows the sequentialization of parallel reductions as in Example 2 (Lemma 12). This assumption is formalized in an axiom *Residuals* that allows a reduction r_1 to inhibit a reduction r_2 of same source term only if the position of one reduction is a prefix of the position of the other, or in other words one redex is a subterm of the other. Thus, virtually any reasonable term-style rewriting system can be represented. We list below some classes of higher-order systems that are rewriting-theoretically interesting, and moreover particularly relevant to the study of functional programming.

Conditional calculi *Closed reduction strategies* [14] are a family of fragments of the λ-calculus particularly convenient for implementation purposes as they are α-conversion free. They introduce conditions on β-reduction based on the free variables of the redexes.

Non-right-linear systems The fixpoint seen above is such a system, since t appears twice in the right member of $\mu x.t \;\rightarrow\; t^{\{x := \mu x.t\}}$.

Non-orthogonal systems Higher-order rewriting is useful to the study of *program transformations*. This may yield non-orthogonal systems as the fully-lazy λ-lifting presented in [7]. Another famous reservoir of non-orthogonal higher-order rewriting systems is the literature on calculi with *explicit substitutions* [1, 20]. Explicit substitutions are an important tool to bridge the gap between the formal definition of β-reduction and its various implementations. Section 5 presents a calculus with `let`-constructs having a comparable flavor.

Non-sequential systems Defining functions by *pattern matching* is a key feature of functional programming. One of its major difficulties is the treatment of matching failure, which is non-sequential in general [18], and similar to the *parallel or* operator.

3 The axiomatic framework

This section defines *Sharing-via-Labelling Systems (SvLS*, Definition 17, Section 3.3*)* and their *parallel labelled reduction* (Definition 18), and then proves that the latter preserves the sharing property (Theorem 21).

The construction follows the principles presented above in three steps: *Abstract Term Rewriting Systems (ATRS*, Definition 9, Section 3.2*)* are the basic formalism for term rewriting. *Marked ATRS (*Definition 13, Section 3.2*)* enrich *ATRS* with a marking of the potential effects of a reduction (the effect zone). *SvLS* are *Marked ATRS* whose terms are partially labelled. We start by some basic and mostly natural definitions on terms.

3.1 Basic syntax

▶ **Definition 3** (Terms). Let $\mathcal{S}$ be a finite or countable set called **signature**. **Terms** over $\mathcal{S}$ are given by the following grammar:

$$t \quad ::= \quad f \mid f(t_1, ..., t_n)$$

where $f \in \mathcal{S}$ and $n \geq 1$. Remark that the symbols in $\mathcal{S}$ have *a priori* variable arities. This can be constrained later. A **context** c is a term with a hole. Write _ for the hole, and $c[t]$ for the term obtained by replacing the hole of c by the term t. ◀

▶ **Example 4.** $@(\lambda x(\_), y)[@(x, y)] = @(\lambda x(@(x, y)), y)$ ◀

▶ **Definition 5** (Positions). A **position** p is a possibly empty sequence of positive integers. Write ϵ the **empty position**, and $p \cdot q$ the **concatenation** of two positions p and q. The concatenation extends to sets as follows: $P \cdot Q = \{p \cdot q \mid p \in P,\ q \in Q\}$. Write $p \cdot Q$ as a shorthand for $\{p\} \cdot Q$. A position p is a **prefix** of a position q, written $p \prec q$, if there is a position p' such that $p \cdot p' = q$. If moreover $p' \neq \epsilon$ then the prefix is **strict**. Two positions p, q are **disjoint** when none is a prefix of the other. An **initial segment** is a set of positions P such that for any positions $q \prec p$, if $p \in P$ then $q \in P$.

The **set of positions** $pos(t)$ of a term t is defined as follows:

$$\begin{array}{rcl} pos(f) & = & \{\epsilon\} \\ pos(f(t_1, ..., t_n)) & = & \{\epsilon\} \cup \bigcup_{1 \leq i \leq n} i \cdot pos(t_i) \end{array}$$

For any term t and position $p \in pos(t)$, write $t(p)$ the **symbol** at position p in t and $t|_p$ the **subterm** of t at position p, defined as follows:

$$\begin{array}{rcl|rcl} f(\epsilon) & = & f & f|_\epsilon & = & f \\ f(t_1, ..., t_n)(\epsilon) & = & f & f(t_1, ..., t_n)|_\epsilon & = & f(t_1, ..., t_n) \\ f(t_1, ..., t_n)(i \cdot p) & = & t_i(p) & f(t_1, ..., t_n)|_{i \cdot p} & = & t_i|_p \end{array}$$ ◀

▶ **Example 6** (λ-terms). Let $\mathcal{X}$ be a countable set of variables. In the signature $\mathcal{S}_\Lambda$ of the λ-calculus we consider the binding λx as one symbol: $\mathcal{S}_\Lambda = \mathcal{X} \cup \{\lambda x \mid x \in \mathcal{X}\} \cup \{@\}$. Let $t = @(\lambda x(@(x, y)), y)$, then $pos(t) = \{\epsilon, 1, 11, 111, 112, 2\}$, $t(1) = \lambda x$ and $t|_{11} = @(x, y)$. ◀

▶ **Definition 7** (Parallel replacement). For any terms t, u and any set of pairwise disjoint positions $P \subseteq pos(t)$, write $t[u]_P$ the **parallel replacement** defined as follows:

$$\begin{array}{rcl} t[u]_\emptyset & = & t \\ t[u]_{\{\epsilon\}} & = & u \\ f(t_1, ..., t_n)[u]_P & = & f(t_1[u]_{P_1}, ..., t_n[u]_{P_n}) \quad \text{where } P = \bigcup_i i \cdot P_i \end{array}$$ ◀

▶ **Example 8** (Parallel replacement). $f(f(a,a),a)[b]_{\{12,2\}} = f(f(a,b),b)$. ◀

3.2 Abstract Term Rewriting Systems (*ATRS*)

We define *Abstract Term Rewriting Systems* as a specialization of *ARS* where the set $\mathcal{O}$ of objects is a set of terms. Each reduction r is associated to a redex $\mathtt{dex}(r)$, a reduct $\mathtt{duct}(r)$, and a context $\mathtt{ctx}(r)$ (Fig. 2).

Axiom *Source & Target* ensures that the source $\mathtt{src}(r)$ and the target $\mathtt{tgt}(r)$ of a reduction r are in $\mathcal{O}$ (which has to be understood as the set of *well-formed* terms). Axiom *Residuals* requires that two reductions concerning disjoint positions do not interfere with each other (Fig. 4). It is formalized using a notion of *equivalence* of redexes characterizing redexes that differ only by their context. Axiom *Residuals* ensures that parallel reduction can be simulated by sequences of single reduction steps (Lemma 12).

▶ **Definition 9** (ATRS). Let $\Sigma = (\mathcal{S}, \mathcal{T}, \mathcal{R}, \mathtt{dex}, \mathtt{duct}, \mathtt{ctx})$ be a tuple where:

- $\mathcal{S}$ is a signature.
- $\mathcal{T}$ is a set of terms over $\mathcal{S}$.
- $\mathcal{R}$ is a countable set whose elements are called **reductions**, and for any $r \in \mathcal{R}$:
  - $\mathtt{dex}(r)$ is a term called **redex** of r.
  - $\mathtt{duct}(r)$ is a term called **reduct** of r.
  - $\mathtt{ctx}(r)$ is a context called **context** of r.

For any $r \in \mathcal{R}$ call **source** (resp. **target**) the term $\mathtt{src}(r) = \mathtt{ctx}(r)[\mathtt{dex}(r)]$ (resp. $\mathtt{tgt}(r) = \mathtt{ctx}(r)[\mathtt{duct}(r)]$) and say $\mathtt{src}(r)$ reduces to $\mathtt{tgt}(r)$. Write $t \to t'$ when t reduces to t' and write $t \twoheadrightarrow t'$ if a sequence of reductions leads from t to t'. For any $r \in \mathcal{R}$ call **root** of r the position $\mathtt{root}(r)$ of the hole of $\mathtt{ctx}(r)$. Two reductions r_1, r_2 are called **equivalent** if $\mathtt{dex}(r_1) = \mathtt{dex}(r_2)$ and $\mathtt{duct}(r_1) = \mathtt{duct}(r_2)$. For any $t \in \mathcal{T}$ and $p \in pos(t)$ write $\mathcal{R}(t,p)$ the set of all the reductions $r \in \mathcal{R}$ such that $\mathtt{src}(r) = t$ and $\mathtt{root}(r) = p$.
Σ is an **Abstract Term Rewriting System** if the two following axioms are satisfied:
Source & Target: For any $r \in \mathcal{R}$, $\mathtt{src}(r) \in \mathcal{T}$ and $\mathtt{tgt}(r) \in \mathcal{T}$.
Residuals: Let $r_1 \in \mathcal{R}$ and $p \in pos(\mathtt{src}(r_1))$ such that $\mathtt{root}(r_1)$ and p are disjoint. Then for any $r_2 \in \mathcal{R}(\mathtt{src}(r_1), p)$ there is $r_2' \in \mathcal{R}(\mathtt{tgt}(r_1), p)$ equivalent to r_2. ◀

In any *ATRS* one can define a notion of parallel reduction that performs several disjoint reductions at one go.

▶ **Definition 10** (Parallel reduction). Let t be a term, and $\{r_1, ..., r_n\}$ a set of reductions of source t whose roots are pairwise disjoint. Write $t' = t[\mathtt{duct}(r_1)]_{\mathtt{root}(r_1)} ... [\mathtt{duct}(r_n)]_{\mathtt{root}(r_n)}$ the term t where each redex has been replaced by its reduct. Say t reduces parallely to t' and write $t \Rightarrow t'$. ◀

▶ **Example 11.** Let r_1 and r_2 be two reductions such that:

$$\begin{array}{rcl|rcl} \mathtt{dex}(r_1) & = & a & \mathtt{src}(r_2) & = & f(f(a,a),a) \\ \mathtt{duct}(r_1) & = & b & \mathtt{tgt}(r_2) & = & f(f(a,a),b) \\ \mathtt{ctx}(r_1) & = & f(f(a,\_),a) & \mathtt{root}(r_2) & = & 2 \end{array}$$

Then $f(f(a,a),a) \Rightarrow f(f(a,b),b)$ by parallel reduction of $\{r_1, r_2\}$. ◀

▶ **Lemma 12** (Simulation of parallel reduction). *In any* ATRS, *if* $t \Rrightarrow t'$ *then* $t \twoheadrightarrow t'$.

Proof. By induction on the number of redexes reduced in parallel, with axiom *Residuals*. ◀

An *ATRS* can be extended with *effect zones* identifying in each reduction a part of the reduct containing all the positions that have possibly been created or modified by the reduction (Fig. 3). Axiom *Effect zone* states this property by contraposition: any subterm of the reduct which does not intersect the effect zone is syntactically equal to a strict subterm of the redex. This forms a *Marked ATRS*.

▶ **Definition 13** (Marked ATRS). Let $\Sigma = (\mathcal{S}, \mathcal{T}, \mathcal{R}, \mathtt{dex}, \mathtt{duct}, \mathtt{ctx}, \mathtt{effz})$ be a tuple where:

- $(\mathcal{S},\mathcal{T},\mathcal{R},\mathtt{dex},\mathtt{duct},\mathtt{ctx})$ is an *ATRS*.
- For any $r \in \mathcal{R}$, $\mathtt{effz}(r)$ is an initial segment of $pos(\mathtt{duct}(r))$, called **effect zone** of r.

Σ is a **Marked** ***ATRS*** if the following axiom is satisfied:

Effect zone: For any $r \in \mathcal{R}$ and $p \in pos(\mathtt{duct}(r)) \setminus \mathtt{effz}(r)$ there is $q \in pos(\mathtt{dex}(r))$ such that $q \neq \epsilon$ and $\mathtt{dex}(r)|_q = \mathtt{duct}(r)|_p$. ◀

The condition $q \neq \epsilon$ ensures that any copy of the root symbol of the redex is in the effect zone, which forces the generation of new labels in the next section. This is required for the preservation of the so-called *independence* property (Definition 19, Lemma 20).

▶ **Example 14.** Let r be a reduction such that $\mathtt{dex}(r) = g(f(a,b))$ and $\mathtt{duct}(r) = h(g(f(a,b)),a)$. Any initial segment of $\{\epsilon, 1, 11, 111, 112, 2\}$ that includes $\{\epsilon, 1\}$ is an admissible effect zone for r. In particular, so are $\{\epsilon, 1\}$, $\{\epsilon, 1, 2\}$, and $\{\epsilon, 1, 11, 111, 112, 2\}$. See Section 4.2 for the impact of the choice of an effect zone. ◀

3.3 Sharing-via-Labelling Systems (*SvLS*)

We define *Sharing-via-Labelling Systems* as a specialization of *Marked ATRS* where the signature contains possibly labelled symbols and where the labels are partially ordered by a contribution relation $\hookrightarrow$.

Axiom *Redex head label* requires the head symbol of the redex of any reduction to be labelled. Axiom *Separation* prevents the labels of an effect zone from clashing. Axiom *Contribution* controls the progression of the labels along $\hookrightarrow$.

▶ **Definition 15** (Labelled signature). Let $\mathcal{S}$ be a signature, and $\mathcal{L}$ be a countable set whose elements are called **labels**. The **labelled signature** $\mathcal{S}[\mathcal{L}]$ is a signature defined by $\mathcal{S}[\mathcal{L}] = \mathcal{S} \cup \{f^\alpha | f \in \mathcal{S}, \alpha \in \mathcal{L}\}$. Let t be a term over a labelled signature $\mathcal{S}[\mathcal{L}]$, and $p \in pos(t)$. If $t(p) = f^\alpha$ with $f \in \mathcal{S}$ and $\alpha \in \mathcal{L}$, then write $\tau_p(t) = \alpha$ the **label** at position p in t. Else $\tau_p(t)$ is undefined. From now on, by writing $\tau_p(t)$ we suppose the label is defined. ◀

The sharing property is the main invariant we want to preserve:

▶ **Definition 16** (Sharing property). A term t on a labelled signature $\mathcal{S}[\mathcal{L}]$ has the **sharing property**, written $\mathbb{S}(t)$, when for any positions $p, q \in pos(t)$ if $\tau_p(t) = \tau_q(t)$ then $t|_p = t|_q$. ◀

▶ **Definition 17** (SvLS). Let $(\mathcal{L}, \hookrightarrow, \Sigma)$ be a tuple where:

- $\mathcal{L}$ is a countable set of labels.
- $\hookrightarrow$ is an irreflexive and transitive relation on $\mathcal{L}$ called **contribution relation**.
- $\Sigma = (\mathcal{S}[\mathcal{L}], \mathcal{T}, \mathcal{R}, \mathtt{dex}, \mathtt{duct}, \mathtt{ctx}, \mathtt{effz})$ is a *Marked ATRS* over a signature $\mathcal{S}[\mathcal{L}]$.

For any t, if there is $p \in pos(t)$ such that $\tau_p(t) = \alpha$ then we say α **occurs** in t. If moreover $t = \mathtt{duct}(r)$ for some $r \in \mathcal{R}$ and $p \in \mathtt{effz}(r)$ then we say α **occurs** in $\mathtt{effz}(r)$.

$(\mathcal{L}, \hookrightarrow, \Sigma)$ is a **Sharing-via-Labelling System** if the following axioms are satisfied:

Redex head label: For any reduction $r \in \mathcal{R}$, the head label $\tau_\epsilon(\mathtt{dex}(r))$ is defined. Write $\tau(r)$ as a shorthand for $\tau_\epsilon(\mathtt{dex}(r))$.

Separation: For any $r \in \mathcal{R}$ such that $\mathbb{S}(\mathtt{dex}(r))$, for any $p, q \in \mathtt{effz}(r)$, if $\tau_p(\mathtt{duct}(r)) = \tau_q(\mathtt{duct}(r))$, then $\mathtt{duct}(r)|_p = \mathtt{duct}(r)|_q$.

Contribution: For any $r \in \mathcal{R}$ and $\beta \in \mathcal{L}$ occurring in $\mathtt{effz}(r)$ one has $\tau(r) \hookrightarrow \beta$. Moreover for any $\alpha \in \mathcal{L}$, if $\alpha \neq \tau(r)$ and $\alpha \hookrightarrow \beta$ then there is $\gamma \in \mathcal{L}$ in $\mathtt{dex}(r)$ such that $\alpha \hookrightarrow \gamma$.

A label $\alpha \in \mathcal{L}$ is **initial** if there is no $\beta \in \mathcal{L}$ such that $\beta \hookrightarrow \alpha$. A term t is **initial** if all its labels are initial and different. ◀

Remark that $\to$ does not preserve the sharing property: only the parallel labelled reduction defined below and the corresponding reduction sequences have a graphical meaning. The parallel labelled reduction simulating graph reduction is the simultaneous reduction of all the redexes with a given label.

▶ **Definition 18** (Parallel labelled reduction)**.** Let t be a term in a *SvLS* such that $\mathbb{S}(t)$, and α be a label in t. Write $P_\alpha = \{q \in pos(t) \mid \tau_q(t) = \alpha\}$. By $\mathbb{S}(t)$ remark that the positions in P_α are pairwise disjoint. Let $r \in \mathcal{R}$ with $\tau(r) = \alpha$ such that for any $p \in P_\alpha$ there is $r_p \in \mathcal{R}(t, p)$ equivalent to r. Then t reduces parallely to $t' = t[\mathtt{duct}(r)]_{P_\alpha}$ (Definition 10), which is written $t \Rightarrow_\alpha t'$ and called parallel labelled reduction. ◀

The proof of the preservation of the sharing property requires an invariant of $\Rightarrow_\alpha$:

▶ **Definition 19** (Independence property)**.** A term t in a *SvLS* has the **independence property**, written $\mathbb{I}(t)$, when there are no two labels α and β in t such that $\alpha \hookrightarrow \beta$. ◀

▶ **Lemma 20** (Independence)**.** *In any* SvLS*, if* $\mathbb{S}(t)$*,* $\mathbb{I}(t)$ *and* $t \Rightarrow_{\alpha_0} t'$*, then* $\mathbb{I}(t')$*.*

Proof. By definition, there is a reduction r such that $\tau(r) = \alpha_0$ and the parallel reduction uses $\mathtt{dex}(r)$ and $\mathtt{duct}(r)$ as unique redex and reduct. Let α, β be two labels in t' s.t. $\alpha \hookrightarrow \beta$.

- If β occurs in t, then by $\mathbb{I}(t)$ there is no occurrence of α in t. Hence α occurs in $\mathtt{effz}(r)$ and by axiom *Contribution* $\alpha_0 \hookrightarrow \alpha$. By transitivity $\alpha_0 \hookrightarrow \beta$, which contradicts $\mathbb{I}(t)$.
- If there is no occurrence of β in t, then β occurs in $\mathtt{effz}(r)$.
  - If α occurs in t, then case on $\alpha \hookrightarrow \beta$ with axiom *Contribution*:
    - Either $\alpha = \alpha_0$, which implies that α_0 occurs in t'. Then by definition of parallel labelled reduction α_0 occurs in $\mathtt{duct}(r)$, and by axiom *Effect zone* α_0 occurs in $\mathtt{effz}(r)$. Hence by axiom *Contribution* $\alpha_0 \hookrightarrow \alpha_0$, which contradicts irreflexivity.
    - Or there is γ in $\mathtt{dex}(r)$ such that $\alpha \hookrightarrow \gamma$, which contradicts $\mathbb{I}(t)$.
  - If there is no occurrence of α in t, then α occurs in $\mathtt{effz}(r)$ and in particular $\alpha_0 \hookrightarrow \alpha$. Moreover $\alpha \neq \alpha_0$, hence by axiom *Contribution* there is γ in $\mathtt{dex}(r)$ s.t. $\alpha \hookrightarrow \gamma$. By transitivity $\alpha_0 \hookrightarrow \gamma$, which contradicts $\mathbb{I}(t)$. ◀

▶ **Theorem 21** (Sharing)**.** *In any* SvLS*, if* $\mathbb{S}(t)$*,* $\mathbb{I}(t)$ *and* $t \Rightarrow_{\alpha_0} t'$*, then* $\mathbb{S}(t')$*.*

Proof. As above there is r with $\tau(r) = \alpha_0$ such that the parallel reduction uses $\mathtt{dex}(r)$ and $\mathtt{duct}(r)$ as unique redex and reduct. Let $p', q' \in pos(t')$. Suppose $\tau_{p'}(t') = \tau_{q'}(t') = \alpha$.

- If none of p', q' is in the effect zone of a reduced redex, then by axiom *Effect zone* there is p (resp. q) in $pos(t)$ such that $t(p) = t'(p')$ (resp. $t(q) = t'(q')$). In particular $\tau_p(t) = \tau_q(t) = \alpha$, and $t|_p = t|_q$ by $\mathbb{S}(t)$. Write $P = \{p_1 \mid \tau_{p_1}(t|_p) = \alpha_0\}$ and $Q = \{q_1 \mid \tau_{q_1}(t|_q) = \alpha_0\}$ and remark that $P = Q$. By definition $t'|_{p'} = (t|_p)[\mathtt{duct}(t)]_P = (t|_p)[\mathtt{duct}(r)]_Q = t'|_{q'}$.
- If both p' and q' are in the effect zone of a reduced redex, then by unicity of $\mathtt{duct}(r)$ and by axiom *Separation* $t'|_{p'} = t'|_{q'}$.

- Else, wlog. p' is in the effect zone of a reduced redex and $\alpha_0 \hookrightarrow \alpha$ (axiom *Contribution*) and q' is not in the effect zone of a reduced redex and α occurs in t (axiom *Effect zone*). This contradicts $\mathbb{I}(t)$. ◀

Remark that any initial term t_i satisfies $\mathbb{S}(t_i)$ and $\mathbb{I}(t_i)$. Then Independence lemma 20 and Sharing theorem 21 have an immediate corollary:

▶ **Corollary 22** (Long-term sharing). *In any* SvLS*, if t_i is an initial term and there is a sequence of parallel labelled reductions from t_i to a term t, then* $\mathbb{S}(t)$*.* ◀

4 First examples

This section presents some simple *Sharing-via-Labelling Systems* and gives useful definitions to handle binders: Subsection 4.1 formalizes the example of Section 2.2, then Subsection 4.2 shows three sharing disciplines for the (unrestricted) λ-calulus.

4.1 The fixpoint operator, formally

Let $\mathcal{X}$ and L be countable sets of variables and initial labels. We consider the signature $\{a, f, g, h\}$ used in Section 2.2. Define:

$$\begin{array}{llcll} \text{Labels } (\mathcal{L}): & \alpha & ::= & \mathfrak{a} \mid [\alpha] \mid [\alpha]\alpha & \mathfrak{a} \in L \\ \text{Terms } (\mathcal{T}_\mu): & t & ::= & x \mid a \mid \mu x^\alpha.t \mid F^\alpha(t_1, ..., t_n) & x \in \mathcal{X},\ F \in \{f, g, h\} \end{array}$$

Free variables $fv(t)$ and bound variables $bv(t)$ of a term t are defined as usual, remembering that $\mu x^\alpha.u$ binds x in u. For any term t and variable x, call **spine** of x in t the initial segment of $pos(t)$ whose elements are the strict prefixes of the positions of the free occurrences of x in t. These are exactly the positions of the subterms of t that would be affected by a substitution of x. By abuse of language, the "labels of a spine" denote the labels at the positions of a spine. The **labelled substitution** $t^{\{x:=\omega,u\}}$ substitutes u for x in t and modifies the labels of the spine of x in t using the label ω. If $x \notin fv(t)$ then $t^{\{x:=\omega,u\}} = t$, else:

$$\begin{array}{rcll} x^{\{x:=\omega,u\}} & = & u & \\ (\mu y^\alpha.t)^{\{x:=\omega,u\}} & = & \mu y^{[\omega]\alpha}.t^{\{x:=\omega,u\}} & y \notin fv(u) \\ (F^\alpha(t_1, ..., t_n))^{\{x:=\omega,u\}} & = & F^{[\omega]\alpha}(t_1^{\{x:=\omega,u\}}, ..., t_n^{\{x:=\omega,u\}}) & F \in \{f, g, h\} \end{array}$$

Remark that this substitution without capture may be undefined in general. See Section 4.2. Define the reduction rule $(\mu)\ :\ \mu x^\omega.t\ \to\ t^{\{x:=\omega,\mu x^{[\omega]}.t\}}$. The corresponding reduction relation is the smallest congruence on terms generated by (μ).

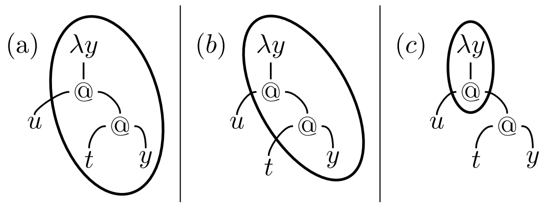
Axiomatic checking The application of the rule (μ) to a term $\mu x^\omega.t$ in any context c defines an obvious reduction r in redex-reduct-context format in a way that satisfies the *ATRS* axioms. Write $t' = t^{\{x:=\omega,\mu x^{[\omega]}.t\}}$. The effect zone of r is the union of the spine of x in t and the occurrences of $\mu x^{[\omega]}$. Hence the positions in t' that are outside of the effect zone are subterms of t, which verifies the axiom *Effect zone*.

Define $\hookrightarrow$ as the least transitive relation such that $\alpha \hookrightarrow [\alpha]$ and $\alpha \hookrightarrow [\alpha]\beta$ for any $\alpha, \beta \in \mathcal{L}$. It is irreflexive. Axiom *Redex head label* is satisfied since all occurrences of μ are labelled. Axiom *Contribution* is satisfied since all labels in the effect zone have the form $[\omega]$ or $[\omega]\alpha$ with ω the label of the redex.

Axiom *Separation*: suppose $\mathbb{S}(\mu x^\omega.t)$, and α occurs twice in the effect zone, at two positions p and q of the reduct. If $\alpha = [\omega]$ then $t'|_p = t'|_q = \mu x^{[\omega]}.t$. Else $\alpha = [\omega]\beta$ for some β occurring at positions p and q in t. Hence, since $\mathbb{S}(t)$ by hypothesis, $t|_p = t|_q$, and $t'|_p = t|_p^{\{x:=\omega,\mu x^{[\omega]}.t\}} = t|_q^{\{x:=\omega,\mu x^{[\omega]}.t\}} = t'|_q$.

4.2 The λ-calculus: β- and α-reduction

We consider the λ-calculus, with the usual notions of free and bound variables, and the notions of spine and labelled substitution adapted from above. We also use L, $\mathcal{L}$ and $\hookrightarrow$ as above, and we label applications and abstractions. Let $(\lambda x.\lambda y.x(ty))u \to \lambda y.u(ty)$ be a reduction where t, u are two terms in which neither x nor y appears free. Three of the possible effect zones are circled below. The effect zone is arbitrary as soon as it contains the spine of x in $\lambda y.x(ty)$, which is circled in case (c).



(a) duplicating the whole body of the function is usual lazy evaluation; (b) avoiding the copy of the free expressions (t here) is Wadsworth's algorithm [29]; (c) duplicating only the spine of x is found in [10].

While these examples have been originally defined with weak reduction strategies, more general strategies are known to allow shorter reductions [17], thus we consider here general β-reduction. As a consequence α-conversion is sometimes needed to avoid blocked terms.

Remark that in our framework only the symbols of the effect zone are allowed to be modified, hence any renaming operation has to be contained in the effect zone. In cases (a) and (b) there is no problem: the effect zone is always large enough. In case (c) there is a solution: add a reduction rule for renaming, with appropriate creations of labels.

$$\text{Renaming:} \qquad \lambda x^{\alpha}.t \quad \to \quad \lambda y^{[\alpha]}.t^{\{x := \alpha, y\}} \qquad y \notin \mathit{fv}(t) \cup \mathit{bv}(t)$$

The effect zone of this reduction is $\epsilon \cdot P$ where P is the spine of x in t. Checking the axioms for this new rule as well as for β-reduction is similar to Section 4.1. Remark that renaming can also be inlined in β, with an appropriate extension of the effect zone.

The need for this new rule and the duplications it implies comes from a new behaviour brought by case (c): sharing-via-labelling allows the sharing of open subterms bound by different binders. Note that this is not allowed in usual higher-order termgraphs [11, 19]. Section 5 develops this point and illustrates how one can take advantage of it.

5 A significant example

This section presents a labelling of a calculus inspired by the *call-by-need λ-calculus* [3]. It emphasizes how the graphs represented by *SvLS* are more general than usual higher-order termgraphs, and takes advantage of this point to present a reduction strategy performing a more-than-fully-lazy sharing.

In the call-by-need λ-calculus $\lambda_{\texttt{let}}$, the β-rule is replaced by the introduction of a `let`-construct that delays the substitution of the argument: $(\lambda x.t)u \to \texttt{let}\ x = u\ \texttt{in}\ t$. From the point of view of the argument u, this keeps only one shared occurrence of u for possibly numerous corresponding occurrences of x in t.

Now, in our context the labels can take care of this sharing already: the substitution of u does not need modifying the labels of u. This safeguard allows us to relax most of the original rules of $\lambda_{\texttt{let}}$ and use the delay introduced by `let` in a new way. Indeed, in $\texttt{let}\ x = u\ \texttt{in}\ t$, from the point of view of t, the `let` makes the value of x unspecified. This allows us to share multiple copies of t bound by different `let`-constructs with different arguments.

Let $\mathcal{X}$, L, $\mathcal{L}$ and $\hookrightarrow$ be as defined above. We define below the labelled terms, where applications and `let`-constructs are labelled (labelling λ-abstractions is possible but not needed). The answers represent λ-abstractions under `let`-constructs. Evaluation focuses are contexts that guide call-by-need shared reduction by pointing one of the redexes to be reduced parallely.

$$\begin{array}{llcll}
\text{Terms } (\mathcal{T}_{\texttt{let}}): & t & ::= & x \mid \lambda x.t \mid @^{\alpha}(t,t) \mid \texttt{let}^{\alpha}\ x = t \texttt{ in } t & x \in \mathcal{X} \\
\text{Answers } (\mathcal{A}_{\texttt{let}}): & A & ::= & \lambda x.t \mid \texttt{let}^{\alpha}\ x = t \texttt{ in } A & x \in \mathcal{X} \\
\text{Eval. focus } (\mathcal{E}_{\texttt{let}}): & E & ::= & \_ \mid @^{\alpha}(E,t) \mid \texttt{let}^{\alpha}\ x = t \texttt{ in } E & x \in \mathcal{X}
\end{array}$$

The application is explicited by a binary constructor @ to ease the reading of labelled terms. Free and bound variables are defined as usual, remembering that both $\lambda x.t$ and $\texttt{let}\ x = u \texttt{ in } t$ bind x in t. Spines and labelled substitutions are adapted from above.

Two main rules (Intro) and (Subst) respectively introduce a `let`-construct to delay the substitution of a β-reduction and perform substitution when the focus of the reduction encounters a variable. The (Scope) rule prevents `let`-constructs from hiding β-redexes. A (GC) rule is added to lighten the examples, and (Renaming) is as discussed in Section 4.2. The system is not orthogonal: (GC) and (Renaming) introduce several critical pairs. The reduction relation is the smallest congruence on terms generated by the rules.

$$\begin{array}{lrcll}
\text{Intro:} & @^{\alpha}(\lambda x.t, u) & \to & \texttt{let}^{[\alpha]}\ x = u \texttt{ in } t & \\
\text{Subst:} & \texttt{let}^{\alpha}\ x = t \texttt{ in } E[x] & \to & \texttt{let}^{[\alpha]}\ x = t \texttt{ in } E[y]^{\{y := \alpha, t\}} & y \text{ fresh} \\
\text{Scope:} & @^{\alpha}(\texttt{let}^{\beta}\ x = t \texttt{ in } A,\ u) & \to & \texttt{let}^{[\alpha]\beta}\ x = t \texttt{ in } @^{[\alpha]}(A,u) & x \notin fv(u) \\
\text{GC:} & \texttt{let}^{\alpha}\ x = t \texttt{ in } u & \to & u & x \notin fv(u) \\
\text{Renaming:} & \texttt{let}^{\alpha}\ x = t \texttt{ in } u & \to & \texttt{let}^{[\alpha]}\ y = t \texttt{ in } u^{\{x := \alpha, y\}} & y \notin fv(u) \cup bv(u)
\end{array}$$

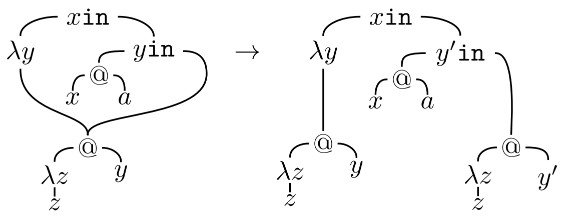
Remark that in rule (Subst) only one occurrence of the variable is substituted and the labels are modified along the substitution.

In the sequence below we follow the underlined subterm $\underline{@^{\delta}(\lambda z.z, y)}$. In usual call-by-need the two copies are separated from step (1), as the value of the first occurrence of x is required. Even with the fully lazy version given in [2], only the duplication of $\lambda z.z$ would be spared, which would not help here. In this example however, the two copies remain shared thanks to the label δ (and later $[\delta]$). They are separated only at the end of step (5). Inbetween, steps (3), (4) and (5) evaluate in parallel the two copies. As a consequence, the function substituted for the second occurrence of x in step (6) is already fully evaluated. The interesting new behaviour observed here is the sharing of two subterms occurring at very different positions: one is buried under a λ-abstraction while the other stands in the evaluation focus.

$$\begin{array}{rll}
 & @^{\alpha}(\ \lambda x.@^{\beta}(x, @^{\gamma}(x,a)),\ \lambda y.\underline{@^{\delta}(\lambda z.z, y)}\ ) & \\
\to_I & \texttt{let}^{[\alpha]}\ x = \lambda y.\underline{@^{\delta}(\lambda z.z, y)} \texttt{ in } @^{\beta}(x, @^{\gamma}(x,a)) & \\
\to_{Su} & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{@^{\delta}(\lambda z.z, y)} \texttt{ in } @^{[[\alpha]]\beta}(\lambda y.\underline{@^{\delta}(\lambda z.z, y)}, @^{\gamma}(x,a)) & (1) \\
\to_I & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{@^{\delta}(\lambda z.z, y)} \texttt{ in let}^{[[[\alpha]]\beta]}\ y = @^{\gamma}(x,a) \texttt{ in } \underline{@^{\delta}(\lambda z.z, y)} & (2) \\
\Rightarrow_I & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{\texttt{let}^{[\delta]}\ z = y \texttt{ in } z} \texttt{ in let}^{[[[\alpha]]\beta]}\ y = @^{\gamma}(x,a) \texttt{ in } \underline{\texttt{let}^{[\delta]}\ z = y \texttt{ in } z} & (3) \\
\Rightarrow_{Su} & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{\texttt{let}^{[\delta]}\ z = y \texttt{ in } y} \texttt{ in let}^{[[[\alpha]]\beta]}\ y = @^{\gamma}(x,a) \texttt{ in } \underline{\texttt{let}^{[\delta]}\ z = y \texttt{ in } y} & (4) \\
\Rightarrow_{GC} & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{y} \texttt{ in let}^{[[[\alpha]]\beta]}\ y = @^{\gamma}(x,a) \texttt{ in } \underline{y} & (5) \\
\to_{GC}^{Su} & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{y} \texttt{ in } @^{\gamma}(x,a) & \\
\to_{GC}^{Su} & @^{[[[\alpha]]\gamma]}(\lambda y.\underline{y})a & (6)
\end{array}$$

The labelled term obtained at step (2) and its corresponding graph are represented below. They show how sharing-via-labelling allows the sharing of open subterms affected by different bindings. This kind of graph is usually called *non-admissible* [29], but in our case the labelled term clearly shows that there is no binding ambiguity, and thus that using this graph is possible. The point is: in this setting α-conversion cannot be silent. Indeed, renaming one occurrence of y requires breaking the sharing and modifying one of the occurrences of the label δ.

$$\begin{array}{rl} & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{@^{\delta}(\lambda z.z, y)} \\ & \texttt{in let}^{[[[\alpha]]\beta]}\ y = @^{\gamma}(x, a) \\ & \quad \texttt{in } \underline{@^{\delta}(\lambda z.z, y)} \\ \\ \to & \texttt{let}^{[[\alpha]]}\ x = \lambda y.\underline{@^{\delta}(\lambda z.z, y)} \\ & \texttt{in let}^{[[[[\alpha]]\beta]]}\ y = @^{\gamma}(x, a) \\ & \quad \texttt{in } \underline{@^{[[[[\alpha]]\beta]]\delta}(\lambda z.z, y)} \end{array}$$



In this case the "physical identity" of one copy of the shared redex $@^{\delta}(\lambda z.z, y)$ is modified, which illustrates the *dynamic* identification mentioned in introduction.

Axiomatic checking Define a reduction as the application of any rule in any context. The axioms of *ATRS* are satisfied. Define the effect zone of a reduction as the set of the positions where the labels are changed. Remark that the collapsing rule (GC) has an empty effect zone, while the effect zones of (Subst.) and (Renaming) correspond to spines. The axiom *Effect zone* is satisfied. The axioms of *SvLS* are satisfied as in Section 4.1.

▶ Remark. The labelled substitution extends implicitly the right-hand-sides of the rules, hence this system does not fit directly in usual (higher-order) term rewriting frameworks. This is fixed as soon as the rule schemes explicitly mention the spines, as it is done in [7]. ◀

6 Related work

Sharing and graph rewriting have already been described in many different ways, some of which are reviewed here. We compare them with our framework along three directions:
Expressive power: the amount of compatible rewriting systems and their features.
Sharing power: the level of sharing achieved on compatible systems.
Formal complexity: the simplicity or complexity of defining and handling the system.

Adressed TRS D. Dougherty et al. [13] propose labelled first-order systems with a similar sharing power. They allow non-orthogonality, and also cycles representing infinite terms, which we do not here. They do not however mention higher-order systems. Their technology is slightly more complex than term rewriting: the new labels are taken globally fresh, thus the reduction of a redex depends on its global context and parallel reduction is not decomposed.

Term graphs Term graphs [9, 27, 19, 8] provide sharing facilities for any first-order or higher-order system, with an equivalent sharing power in many cases, and also allow cycles [8]. They have a far higher formal complexity, since they require subtle correctness criteria and notions of morphisms of graphs with binders. They forbid the sharing of open subterms affected by different binders, which limits the reduction strategies with sharing (Section 5).

Call by need The call-by-need λ-calculus [3] and its fully lazy version [2] give a simple and elegant account of Wadsworth's graph algorithm for the evaluation of the λ-calculus [29]. N. Yoshida [30] also develops similar ideas by using explicit substitutions in place of `let`-constructs. These works also manage to use only terms and one-step term reduction. On the other hand they require structural rules or structural equations. The latter point however has been recently improved in [12]. Concerning the expressive power, these works are a study of the particular case of the λ-calculus, which is orthogonal.

Optimal sharing The optimal implementations of higher-order systems [23, 22, 4, 5] perform a better sharing than our framework, and apply to a large class of orthogonal higher-order systems, through Interaction Systems [5]. They do not apply however to all orthogonal higher-order systems, and do not apply at all to non-orthogonal systems. They also achieve optimal sharing at the cost of a far greater formal complexity.

Interaction nets Interaction nets can implement wide classes of first-order and higher-order rewriting systems [15, 16, 24]. They are known in particular for the suboptimal yet powerful sharing they realize in systems like [24]. The limit of their expressive power is their sequential nature. Also their atomization of every operation makes the link with the term specifications technically very demanding.

7 Conclusion

The axiomatic sharing-via-labelling systems presented in this paper provide a general framework for the sharing of subterms in term rewriting. This very expressive framework applies to higher-order and non-orthogonal rewriting systems and covers a wide range of shared reduction algorithms based on acyclic graphs. The formalism is simpler than most of the other approaches to shared rewriting since it is definable within the realm of term rewriting.

The framework generalizes the previous label-based approaches to sharing inspired by optimality theory [25, 10, 6, 7] by dropping the causal content of the labels. This makes it possible to eliminate their restriction to weak reduction as well as their need for indirection nodes, and to define more efficient shared reduction strategies. Section 5 in particular shows this, illustrating the power of ingredients as simple as term rewriting and acyclic sharing.

This work opens new perspectives for rewriting theory and functional programming:

Study of general term rewriting The *Abstract Term Rewriting Systems (ATRS)* are an intermediate formalism between fully abstract rewriting frameworks and concrete term rewriting frameworks. This granularity makes it possible to grasp specific properties of term rewriting that are common to all term frameworks.

Specification of graph rewriting Sharing-via-labelling provides a simple approach to graph rewriting, which is now applicable on a large scale.

Analysis of functional programming Our work provide a homogeneous framework for comparing shared implementations of functional programming languages, in the line of [7].

Implementation of functional programming Sharing-via-labelling provides means to imagine and express new graph algorithms and new reduction strategies for the implementation of functional programming languages, as illustrated in Section 5.

Acknowledgements I would like to thank Delia Kesner for her good advice and constant support, as well as the anonymous referees for their helpful comments and suggestions.

References

1 M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991.

2 Z.M. Ariola and M. Felleisen. The call-by-need lambda calculus. *J. Funct. Program.*, 7(3):265–301, 1997.

3 Z.M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. In *POPL*, pages 233–246, 1995.

4 A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages.* Cambridge University Press, 1998.

5 A. Asperti and C. Laneve. Interaction systems ii: The practice of optimal reductions. *TCS*, 159(2):191–244, 1996.

6 T. Balabonski. Optimality for dynamic patterns. In *PPDP'10*, pages 231–242, 2010.

7 T. Balabonski. A unified approach to fully lazy sharing. In *POPL'12*, 2012.

8 P. Baldan, C. Bertolissi, H. Cirstea, and C. Kirchner. A rewriting calculus for cyclic higher-order term graphs. *Mathematical Structures in Computer Science*, 17(3):363–406, 2007.

9 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 (2)*, pages 141–158, 1987.

10 T. Blanc, J.-J. Lévy, and L. Maranget. Sharing in the Weak Lambda-Calculus Revisited. In *Reflections on Type Theory, Lambda Calculus and the Mind*, 2007.

11 S. Blom. *Term Graph Rewriting - Syntax and Semantics.* Ph.D. thesis, 2001.

12 S. Chang and M. Felleisen. The call-by-need lambda calculus, revisited. In *ESOP*, 2012.

13 D. Dougherty, P. Lescanne, L. Liquori, and F. Lang. Addressed Term Rewriting Systems: Syntax, Semantics, and Pragmatics: Extended Abstract. *ENTCS*, 127(5):57–82, 2005.

14 M. Fernández, I.Mackie, and F.-R. Sinot. Closed reduction: explicit substitutions without alpha-conversion. *MSCS*, 15(2):343–381, 2005.

15 M. Fernández and I. Mackie. Interaction nets and term-rewriting systems. *TCS*, 190(1):3–39, 1998.

16 M. Fernández, I. Mackie, S. Sato, and M. Walker. Recursive functions with pattern matching in interaction nets. *ENTCS*, 253(4):55–71, 2009.

17 C.K. Holst and D.K. Gomard. Partial evaluation is fuller laziness. *SIGPLAN Not.*, 26(9):223–233, 1991.

18 C.B. Jay and D. Kesner. First-class patterns. *J. Funct. Program.*, 19(2):191–225, 2009.

19 W. Kahl. Relational treatment of term graphs with bound variables. *Logic Journal of the IGPL*, 6(2):259–303, 1998.

20 D. Kesner. The theory of calculi with explicit substitutions revisited. In *CSL*, pages 238–252, 2007.

21 U. Dal Lago and S. Martini. On constructor rewrite systems and the lambda-calculus. In *ICALP'09*, pages 163–174, 2009.

22 J. Lamping. An algorithm for optimal lambda calculus reduction. In *POPL*, 1990.

23 J.-J. Lévy. Optimal reductions in the lambda-calculus. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalisms*, pages 159–191, 1980.

24 I. Mackie. Efficient lambda-evaluation with interaction nets. In *RTA*, pages 155–169, 2004.

25 L. Maranget. Optimal Derivations in Weak Lambda-calculi and in Orthogonal Terms Rewriting Systems. In *POPL'91*, pages 255–269, 1991.

26 P.-A. Melliès. *Description Abstraite des Systèmes de Réécriture.* Ph.D. thesis, 1996.

27 D. Plump. Term graph rewriting. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 2, pages 3–61, 1999.

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

29 C. P. Wadsworth. *Semantics and Pragmatics of the Lambda Calculus.* Ph.D. thesis, 1971.

30 N. Yoshida. Optimal reduction in weak-λ-calculus with shared environments. *J. of Computer Software*, 11(5):2–20, 1994.

On the Decidability Status of Reachability and Coverability in Graph Transformation Systems*

Nathalie Bertrand[1], Giorgio Delzanno[2], Barbara König[3], Arnaud Sangnier[4], and Jan Stückrath[3]

1 **Inria Rennes Bretagne Atlantique, France**
2 **Università di Genova, Italy**
3 **Universität Duisburg-Essen, Germany**
4 **LIAFA, Univ Paris Diderot, Sorbonne Paris Cité, CNRS, France**

Abstract

We study decidability issues for reachability problems in graph transformation systems, a powerful infinite-state model. For a fixed initial configuration, we consider reachability of an entirely specified configuration and of a configuration that satisfies a given pattern (coverability). The former is a fundamental problem for any computational model, the latter is strictly related to verification of safety properties in which the pattern specifies an infinite set of bad configurations. In this paper we reformulate results obtained, e.g., for context-free graph grammars and concurrency models, such as Petri nets, in the more general setting of graph transformation systems and study new results for classes of models obtained by adding constraints on the form of reduction rules.

1998 ACM Subject Classification D.2.2 Design Tools and Techniques, D.2.4 Software/Program Verification, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases decidability, reachability, graph transformation systems

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.101

Category Regular Research Paper

1 Introduction

Graph transformation systems (GTS) form an intuitive, but precise modelling framework, which has been extensively studied since its introduction in the 1970's. A graph transformation system consists of an initial graph and a set of reduction rules, which rewrite graphs and thus generate a transition system with graphs as states. Applications come from diverse areas such as the specification of UML model transformation [14] or encoding of process calculi [4].

In the past there has been a strong focus on questions of (categorical) semantics and expressiveness, while algorithmic issues received much less attention. Especially, different from related formalisms such as Petri nets [15], there is no systematic study of decidability results for graph transformation systems.

Taking inspirations from recent work on concurrency models [5, 8, 19, 20, 23], we consider here decidability issues for two fundamental problems: *reachability* and *coverability* of a given graph G_f from an *initial graph* G_0. The first problem requires the existence of a

* Research partially supported by DFG project GaReV.

computation from G_0 to (a graph isomorphic to) G_f. The second one requires the existence of a computation from G_0 to some graph G that includes G_f as a subgraph.

While it is straightforward to determine the decidability status of these problems for general graph transformation systems, where the problems are both undecidable, and finite-state graph transformations, where they are both decidable, there are several other classes of infinite-state systems for which the question can be naturally asked. In this paper we systematically analyse reachability and coverability for several subclasses of GTS obtained as syntactic restrictions of reduction rules.

For special classes of GTS in which rules never change the underlying structure, we also consider an existential formulation of the coverability problem in which the initial configuration is an unknown variable to be discovered. This problem naturally models parameterized verification questions for concurrent systems with a static topology [9].

With our analysis, we combine within a unified framework both known results coming from fields such as context-free graph grammars, concurrency and verification, with new ones that we obtain by considering restrictions such as non-deletion of nodes, well-structuredness of the rewriting relation, and relabelling reductions only. An algorithmic view of graph transformation systems could open new research directions in the field of verification of infinite-state systems.

Due to length restrictions this paper contains mostly proof sketches. A version of this paper including all proofs in full length is published as a technical report [3].

2 What is a Graph Transformation System?

A graph transformation systems (GTS) is defined by a collection of reduction rules that can be used to dynamically modify the structure of an initial hypergraph. Reduction rules can conveniently be defined as graph morphisms. To formalize these ideas, we next define (labelled) hypergraphs – in the following simply called graphs – and graph morphisms.

▶ **Definition 1.** Let Λ be a finite sets of edge labels and $ar\colon \Lambda \to \mathbb{N}_0$ a function that assigns an arity to each label. A *$(\Lambda$-$)$hypergraph* is a tuple (V_G, E_G, c_G, l_G^E) where V_G is a finite set of nodes, E_G is a finite set of edges, $c_G\colon E_G \to V_G^*$ is a connection function and $l_G^E\colon E_G \to \Lambda$ is an edge labelling function. We require that $|c_G(e)| = ar(l_G^E(e))$ for each edge $e \in E_G$

An edge e is called *adjacent* to a node v if v occurs in $c_G(e)$.

We remark here that we consider graphs in which hyperedges have a fixed arity determined by their label. Directed labelled graphs are a special case of hypergraphs where every edge label has arity 2 and every sequence $c_G(e)$ is of length two.

▶ **Definition 2.** Let G, G' be $(\Lambda$-$)$hypergraphs. A *partial hypergraph morphism* (or simply *morphism*) $\varphi\colon G \rightharpoonup G'$ consists of a pair of partial functions $(\varphi_V : V_G \rightharpoonup V_{G'}, \varphi_E : E_G \rightharpoonup E_{G'})$ such that for every $e \in E_G$ it holds that $\varphi_V(c_G(e)) = c_{G'}(\varphi_E(e))$ whenever $\varphi_E(e)$ is defined. Furthermore if a morphism is defined on an edge, it must be defined on all nodes adjacent to it.

A morphism is called *label-preserving* if in addition $l_G^E(e) = l_{G'}^E(\varphi_E(e))$ for all $e \in E_G$, where $\varphi_E(e)$ is defined. *Total morphisms* are denoted by an arrow of the form $\to$.

In the following we drop the subscripts and write φ instead of φ_V and φ_E. Note that in the literature graph morphisms are usually label-preserving. Here we are more flexible in our definition, since we want to use graph morphisms also in order to define relabellings.

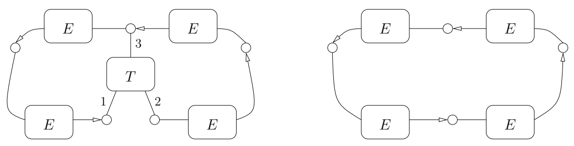


Figure 1 A graph (left) and its minor (right).

Isomorphism, Subgraphs, and Minors

Two graphs G, H are called *isomorphic* if there is a bijective, label-preserving morphism from G to H. In the rest of the paper we consider the following graph orderings over hypergraphs: $G \sqsubseteq_s H$, if G is a *subgraph* of H, namely, there exists a total, injective and label-preserving morphism from G to H, and $G \sqsubseteq_m H$ if G is a *minor* of H, i.e., G can be obtained from H by (iterative) node deletion, edge deletion and edge contraction. *Edge contraction* for hypergraphs means that the edge is deleted and its adjacent nodes are merged arbitrarily, i.e. merging nodes according to any partition on the adjacent nodes will result in a valid edge contraction (see also [22]). Fig. 1 shows an example of an edge contraction: the 3-ary hyperedge labelled T (where the nodes attached to the edge are numbered 1, 2, 3 in their respective order) is contracted with partition $\{1,2\}$, $\{3\}$, which means that we obtain a circle of E-edges.

Based on results in [22] it can be shown that the minor order defined above is a well-quasi order[1] on the set of all hypergraphs. The minor order on hypergraphs was first differently introduced in [17], contracting *all* nodes attached to a hyperedge. However, unfortunately it does not follow from the results in [22] that the order studied in [17] is a wqo. By using the minor order defined above instead, the results of [17] can be recovered, see [18] for a corrected version. Furthermore the subgraph order is a wqo on a restricted set of graphs (for more details see Section 3.5). Note that if G is a subgraph of H, it is also a minor, but not necessarily vice versa.

Graph Rewriting

We now define the rewriting mechanism, a slight extension of single-pushout rewriting (SPO) [13]. One of the most important features of SPO is the handling of so-called "dangling" edges: whenever a node is deleted, all edges attached to it have to be removed as well, also those which are not explicitly deleted. Different from standard SPO we allow as rules partial morphisms which are not necessarily label-preserving. In such a case the corresponding edge is preserved and relabelled. In the definition below this is simulated by removing the old edge and adding a new edge with the modified label. This will be especially important in Section 3.7 where we consider node and edge relabelling as well.

▶ **Definition 3.** A *rewriting rule* is a partial morphism $r\colon L \rightharpoonup R$, where L is called left-hand side and R right-hand side. A *match* (of r) is a total, injective and label-preserving morphism $m\colon L \to G$.[2]

[1] A quasi order (on graphs) is a well-quasi order (wqo) if in every infinite sequence $G_1, G_2, \ldots$ of graphs there are indices $i < j$ with $G_i \sqsubseteq G_j$.

[2] Since m is injective we can assume that it acts as the identity on nodes and edges on which it is defined, i.e., $m(v) = v$ and $m(e) = e$ for $v \in V_L, e \in E_L$. In addition we assume that the set of nodes and edges in R not in the image of r are disjoint from the set of nodes and edges of G.

Given a rule r and a match m, a *rewriting step* or an application of the rule to the graph G, results in a graph H (symbolically $G \Rightarrow_{\mathcal{R}} H$), which is defined as follows. Let E_D be the set of edges $e \in E_G$ which are adjacent to a node $m(v)$ where $r(v)$ is undefined (so-called *dangling edges*). We define the node and edge sets of H as follows: $V_H = V_G \backslash V_L \cup V_R$, $E_H = E_G \backslash (E_L \cup E_D) \cup E_R$.

Define a mapping $\tilde{r}\colon V_G \rightharpoonup V_H$ such that $\tilde{r}(v) = v$ if $v \in V_G \backslash V_L$, and $\tilde{r}(v) = r(v)$ if $v \in V_L$. Finally, define the attachment and edge labelling functions of H:

$$c_H(e) = \begin{cases} c_R(e) & \text{if } e \in E_R \\ \tilde{r}(c_G(e)) & \text{if } e \in E_G \backslash (E_L \cup E_D) \end{cases} \qquad l_H^E(e) = \begin{cases} l_R^E(e) & \text{if } e \in E_R \\ l_G^E(e) & \text{if } e \in E_G \backslash (E_L \cup E_D) \end{cases}$$

Intuitively, we can think of this as follows: L is a subgraph of G, all items of L whose image is undefined under r are deleted, the new items of R are added, merged and connected as specified by r. Whenever a node is deleted, all adjacent edges will be deleted as well. In addition, edges are relabelled as specified by r.

Note that for Definition 3 it would be sufficient to define r solely on nodes and to create and recreate edges instead of preserving them. However, to be consistent with later sections, especially Section 3.7, we allow that r is also defined on edges.

▶ **Example 4.** In the following we introduce an (erroneous) termination detection protocol on a ring, modelled as a GTS to illustrate a rewriting step as well as the differences between the classes of GTS studied in later sections.

The protocol is initialised with a ring structure containing an active process, a passive process and a passive detector as shown in Figure 2. The first three rules presented in Figure 3 allow normal and detector processes to deactivate themselves (3a), activate other processes (3b) and generate new processes (3c). The last three rules shown in Figure 3 allow detector processes to generate termination messages (3d), normal processes to forward these messages (3e) and detector processes to generate a termination flag after receiving a termination message (3f).

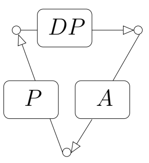
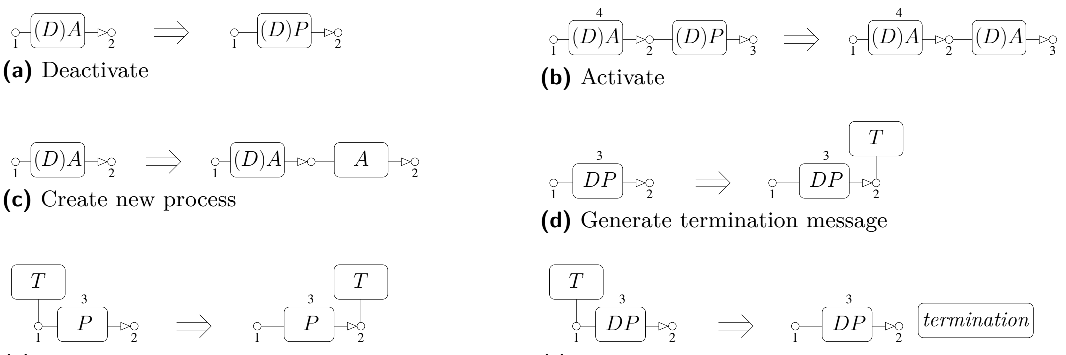


Figure 2 Initial graph.

Edges are represented as boxes with rounded edges and binary edges are drawn as directed edges, indicating the order of the nodes wrt. the edge. The example includes 0-ary hyperedges (with label *termination*), unary hyperedges (with label T) and binary hyperedges with labels (A, P, DA, DP). The label $(D)A$ thereby represents both the label A and DA (label $(D)P$ is uses analogously), i.e. the rule in Figure 3a can either rewrite an A-edge to a



(a) Deactivate

(b) Activate

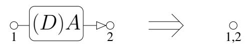
(c) Create new process

(d) Generate termination message

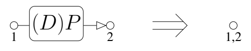
(e) Forward termination message

(f) Termination detection

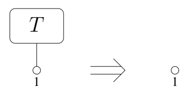
Figure 3 The basic rules of a termination detection protocol.



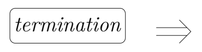
(a) Active process leaves



(b) Passive process leaves



(c) Message lost



(d) Termination flag lost

Figure 4 Additional rules simulating a lossy system.

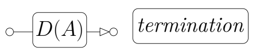


Figure 5 The error graph.

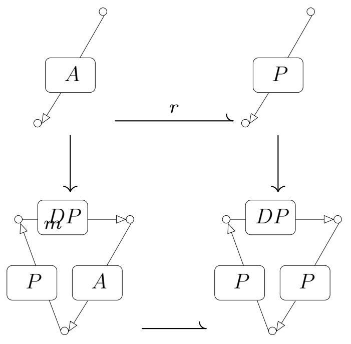


Figure 6 A rewriting example.

P-edge or a DA-edge to a DP-edge. In a rule, an item (node or edge) in the left-hand side numbered i is mapped to the corresponding item on the right-hand side.

This protocol is supposed to detect whether all processes are passive and generate a termination flag in that case. However, by means of the methods of Section 3.5 it can be proven erroneous. Figure 5 represents the pattern which a correct protocol will never produce, that is an active process and a termination flag. Hence we can refute the correctness by showing that a graph containing the pattern is reachable.

In order to model a lossy system, we add rules such that active (4a) and passive (4b) processes can leave the ring and the termination message (4c) and flag (4d) may be lost.

Figure 6 exemplarily shows a rewriting step where the deactivation rule is applied to the initial graph. The active process is thereby deleted and replaced by an identical but passive process, resulting in the bottom right graph, where all processes are passive.

3 An Algorithmic Study of Reachability and Coverability

As mentioned in the introduction, we focus our attention on fundamental decision problems for studying computational properties of a model, namely reachability and coverability. The following definitions are given modulo isomorphism.

▶ **Definition 5.** Given a finite set of rewriting rules $\mathcal{R}$, an initial graph G_0 and a final graph G_f, the Reachability Problem is defined as follows: does $G_0 \Rightarrow^*_{\mathcal{R}} G_f$ hold?

▶ **Definition 6.** Given a finite set of rewriting rules $\mathcal{R}$, an initial graph G_0 and a final graph G_f, the Coverability Problem is defined as follows: is there a graph H such that $G_0 \Rightarrow^*_{\mathcal{R}} H$ and $G_f \sqsubseteq_s H$?

We will now study decidability and undecidability results by extending existing results (e.g., for context-free grammars, GTS and Petri nets, and well-structured transition systems) with new results for several fragments of GTS. The conclusion (Section 5) gives an overview over the various cases in a single diagram. We first recall that the reachability and the coverability problem are both undecidable in the general case. This is a known result. It is quite straightforward to encode Turing machines into graph rewriting and either problem admits a reduction from the halting problem.

Another immediate results follows from restricting the set of reachable graphs to be finite. If it is known that only finitely many graphs up to isomorphism are reachable from G_0, the reachability and coverability problems are decidable for the given GTS. Indeed we just have to enumerate all graphs up to isomorphism and check whether they are isomorphic or in relation to G_f. While the result follows quite trivially, an efficient implementation is more demanding (see for instance the GROOVE tool [21]).

3.1 Symbolic Backward Reachability Algorithm

Symbolic backward analysis turns out to be a convenient tool to answer coverability queries. We present here a generic variant that we apply in some of the proofs in the rest of the paper.

The key idea is to use a graph H as a symbolic representation of its upward closure wrt. some subsumption relation $\sqsubseteq$, namely the infinite set of graphs $up(H) = \{G \mid H \sqsubseteq G\}$. The subsumption relation is in our case either the subgraph inclusion ($G \sqsubseteq_s H$) or the graph minor ordering ($G \sqsubseteq_m H$). For an upward closed set X, we define $\lfloor X \rfloor$ to be a minimal subset of X such that $up(X) = up(\lfloor X \rfloor)$. We can then use $\lfloor X \rfloor$ to describe X sufficiently, as seen below. To check whether a graphs G_f is coverable using the set of rules $\mathcal{R}$, we compute the following sequence: $Cover_0 = \{G_f\}$, $Cover_{i+1} = \left\lfloor Cover_i \cup \bigcup_{r\in\mathcal{R}} Pre_r(Cover_i)\right\rfloor$. The sequence $Cover_0, Cover_1, \ldots$ resembles an exploration which starts from G_f and for each rule r, it applies the rule backwards to the current set of graphs by taking into account their denotation. The operator Pre_r is such that $G \in Pre_r(H)$ iff $\exists H'.\ H \sqsubseteq H',\ G \Rightarrow_r H'$. Hence in the case of subgraph inclusion, by adding auxiliary nodes and edges, we first expand H into graph H', and then we search for a possible match with the right-hand side R of rule r. If such a match exists, we apply a rewriting step backwards replacing R by L. The graph G_f is then coverable if there is an $H \in Cover_i$ with $H \sqsubseteq G_0$ for some $i \in \mathbb{N}_0$.

It is sufficient to compute a minimal set $MPre_r(H)$ of graphs that denotes the infinite set $Pre_r(H)$, i.e. $MPre_r(H) = \lfloor Pre_r(H) \rfloor$. In the case of subgraph inclusion this can be done by considering only expanded graphs H' that are obtained by merging H with subgraphs of R. Indeed we observe that if r is applied backwards to a subgraph of H' that does not overlap with H, then the resulting graph will still contain H as a subgraph, and will never be added (it is subsumed and does not bring any new information).

For the case of the minor ordering we can use similar ideas for the backwards step (see also [17]).

In general the sequence $Cover_0, Cover_1, \ldots$ need not become stationary, hence the procedure need not terminate, but we can show that it terminates for our specific cases. The termination depends mainly on the fact that the subsumption relation is a wqo on the used set of graphs, but may also be affected by the rule set.

3.2 Context-free Graph Transformation Systems

A well-known subclass of graph transformation systems are context-free graph grammars or hyperedge replacement graph grammars [11], where the left-hand side of a rule consists of a single hyperedge and no nodes are deleted or fused. Such grammars are usually used as a language-generating device: labels are partitioned into terminal and non-terminal labels. Only graphs which have only terminal labels are elements of the language. Furthermore G_0 consists of a single edge with a non-terminal label (the axiom) and the same is true for all left-hand sides. Since the distinction between terminal and non-terminal labels does not play a fundamental role for decidability questions, we will drop it in the following.

▶ **Definition 7.** A set $\mathcal{R}$ of rewriting rules is called *context-free* if every rule $r \in \mathcal{R}$ with $r\colon L \rightharpoonup R$ satisfies the following restrictions:

- L has the form $L = (\{v_1, \dots, v_n\}, \{e\}, [e \mapsto v_1 \dots v_n], l_L^E)$, i.e., L consists of a single hyperedge, which is connected to a duplicate-free sequence of nodes.
- r is defined and injective on $v_1, \dots, v_n$. Furthermore r is undefined on e.

Although not context-free, the termination detection example in Section 2 contains context-free rules, e.g. the deactivation rule (3a), the creation rule (3c) and the rule generating a termination message (3f).

▶ **Proposition 8** ([11]). *The reachability problem for context-free graph transformation systems is* NP*-complete. More precisely: there are context-free grammars for which the membership problem is* NP*-complete (in the size of G_f).*

▶ **Proposition 9.** *The coverability problem for context-free graph transformation systems is decidable.*

Proof sketch. The decision procedure is based on the backward exploration algorithm in Section 3.1, where we instantiate the predicate "G subsumes H" by $G \sqsubseteq_s H$ (G is a subgraph of H). Indeed, the key idea is to use a graph H as a symbolic representation of its upward closure wrt. $\sqsubseteq_s$. For the case of context-free rules, termination is obtained by showing that the number of hyperedges occurring in new graphs added at each iteration never increases. From this property and since the arity of hyperedges is finite, we have that the state space we need to explore to search for a graph that subsumes G_0 is always finite (but potentially exponential in the input). More details can be found in [3]. ◀

3.3 Restrictions on Node Deletion and Creation

In many cases graph transformation systems can be viewed as a variant of Petri nets with additional structure. In this case the graph transformation system can be translated into a low-level Petri net and we obtain decidability results from decidability results for Petri nets, possibly with reset and transfer arcs. This is usually the case when we impose some restrictions on the number of nodes that are deleted and/or created, since the main feature that gives GTS more expressiveness than Petri nets is the creation of new nodes. We can obtain such a GTS from the example of Section 2 by deleting the rule which creates new processes and the lossy system rules. To model the lossy system rules transfer arcs are required. This section relies on the intuitions and results of [2], which spells out the connection between SPO and nets with reset arcs. The type of graph transformation systems that are more or less equivalent to Petri nets can be specified as follows. Intuively the rules do not allow node deletion, creation or fusion.

▶ **Proposition 10.** *Assume that the set $\mathcal{R}$ of rewriting rules satisfies the restriction that every rule morphism $r\colon L \rightharpoonup R$ is a bijection on nodes. Then the reachability problem and the coverability problem are decidable.*

Proof sketch. Let G be the initial graph. In this setting V_G remains unchanged by any graph rewriting step and only the edges may change. We construct a Petri net from the GTS to reduce reachability and coverability to reachability and coverability on Petri nets. The places of the Petri net are defined as $P = \{(\ell, s) \in \Lambda \times V_G^* \mid ar(\ell) = |s|\}$. A token in a place (ℓ, s) represents an edge e with $l(e) = \ell$ and $c(e) = s$. The initial graph can be transformed straightforwardly into a marking of the Petri net. The rewriting steps are then simulated by adding a set of transitions for each rule $r\colon L \rightharpoonup R$, taking into account every possible match of r to any possible graph with $|V_G|$ nodes. We achieve this by enumerating all possible mappings h of nodes of L to nodes of V_G, adding one transition for each. For each edge e of L the transition has one input place $(l(e), h(c(e)))$. Analogously the transition has one output place for every edge in R using the same mapping. Effectively a transition takes tokens out of places representing the edges deleted by r and adds tokens in places representing the edges created by r. Since coverability and reachability are decidable for P/T nets, we obtain the same for this variant of GTS. ◀

If we allow node fusion and node deletion and hence also deletion of adjacent dangling edges, but recreate the same number of nodes, we are equivalent in expressivity to Petri nets with transfer arcs.

▶ **Proposition 11.** *Assume that the set $\mathcal{R}$ of rewriting rules satisfies the restriction that for every rule (partial) morphism $r\colon L \rightharpoonup R$ we have $|V_L| = |V_R|$. Then the reachability problem is undecidable, but the coverability problem is decidable.*

Proof sketch. Since the number of nodes of a graph stays constant during rewriting, the encoding of GTS into Petri nets is similar to the proof of Proposition 10. In order to deal with partial morphisms (i.e. node deletion) and non-injective ones (i.e. node fusion), we introduce transitions with transfer arcs that can transfer all tokens contained in a given set of places into a specific place. Reset arcs [12] are a special case in which the transferred tokens are moved to a sink place. Node deletion and subsequent recreation can be simulated via reset arcs, which empty all places corresponding to edges adjacent to a deleted node. Similarly, node fusion can be simulated by transfer arcs which merge the contents of all places corresponding to edges adjacent to nodes that have the same image. Hence we can encode all GTS conforming to the restrictions into transfer nets, inheriting the decidability result from coverability of transfer nets. On the other hand, every reset net can be encoded into a GTS with the above restrictions (see [2]). Hence reachability is undecidable for this class of GTS. ◀

3.4 Non-Deleting Graph Transformation Systems

Now consider GTS that are *non-deleting* (neither edges nor nodes) and in addition label-preserving, i.e., the rule morphism $r\colon L \rightarrowtail R$ is a (total) label-preserving injection. For instance rule (3d) in Section 2 is non-deleting.

▶ **Proposition 12.** *The reachability problem is decidable for non-deleting graph transformation systems.*

Proof sketch. In this setting reachability is decidable, due to the monotonicity of the rules. We apply all rules to derive all possible graphs and stop the derivation if a graph larger than G_f is reached. ◀

However the monotonicity has no effect on coverability and we will show that coverability is in fact undecidable.

▶ **Proposition 13.** *The coverability problem is undecidable for non-deleting graph transformation systems.*

Proof sketch. The coverability problem is undecidable for general GTSs because the halting problem for Turing machines can be reduced to it. This reduction is still possible for non-deleting GTSs. We use a directed path of labeled edges to represent the tape of the Turing machine and add one edge to mark the current state and head position. The input word forms the initial graph and non-deterministically additional blanks are added at both tape ends. Then for each step of the Turing machine, non-deleting rules copy the old tape with the appropriate changes and connect it to the old tape, such that the full Turing machine computation results in a grid-like graph. We have to take into account that without application conditions any rule can be applied an arbitrary number of times using the same matching, leading to a "branching" in the grid. However, we can show that this is not a problem, because different branchings do not interact. Hence, the Turing machine terminates if and only if an edge labeled with a final state is coverable in the GTS. The full proof can be found in [3]. ◀

3.5 Well-Structured Graph Transformation Systems

A good source for decidability results for the coverability problem are well-structured transition systems [16, 1]. For this we need a well-quasi order $\sqsubseteq$, possibly not on the class of all graphs, but on restricted classes. Furthermore it has to be shown that the well-quasi order is a (weak) simulation for the reduction relation, i.e., if $G \sqsubseteq H$ and G is rewritten to G', then H can be rewritten (possibly in several steps) to H' such that $G' \sqsubseteq H'$.

These conditions ensure that every upward-closed set can be finitely represented and that the set of predecessors of an upward-closed set is also upward-closed. If in addition the order is decidable and the direct predecessors can be effectively computed, one automatically obtains a backward search algorithm which can decide coverability.

Here we reuse results of [20, 17] and adapt them to our present setting (hypergraphs and SPO with injective matches as a rewriting formalism). The following decidability results then hold.

▶ **Proposition 14.** *If the set of rules contains edge contraction rules for each edge label (i.e., a rule deleting that edge and merging some of its adjacent nodes using any partition on the node set), then the coverability problem is decidable.*

Proof sketch. We prove well-structuredness of GTS with contraction rules for all possible edge labels. We first recall that the graph minor ordering is a decidable well-quasi ordering [22]. Furthermore, the transition system of a GTS with contraction rules is monotone wrt. the minor ordering. Based on these properties, we can apply the symbolic backward reachability algorithm in Section 3.1, by instantiating the *subsumes* relation with the minor relation, and by taking as $MPre_r$ the computation of the predecessors described in [17] for conflict-free matches, adapted here to injective matches. The fact that the minor ordering is a wqo ensures termination of the predecessor computation and thus of the entire decision procedure. Note also that coverability with subgraph inclusion can be easily reduced to coverability wrt. minor ordering by adding a rule that detects the subgraph G_f and adds an edge with a unused label. Then we check whether this new edge is the minor of a reachable graph. ◀

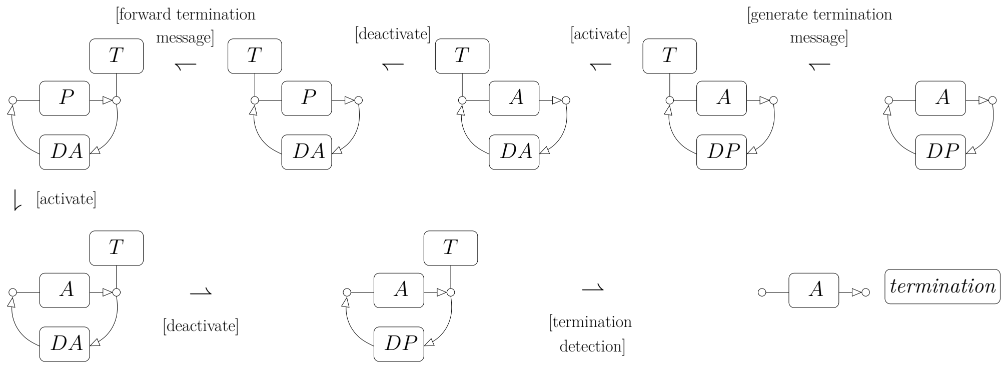


Figure 7 Illustration of a backward search using the GTS of Example 4.

The GTS of Example 4 satisfies the conditions of Proposition 14 because of the lossy system rules (Figure 4). Hence the presented backward search can be used to automatically show that the protocol is erroneous, as shown in Figure 7. Beginning with the unwanted pattern, i.e. an active process and the termination flag, the rules are applied backwards until ultimately reaching a minor of the initial graph. In fact the result is a set of minimal graphs describing all initial graphs for which the protocol is erroneous. Note that in the first step the graph first has to be extended, adding the passive detector, before the rule can be applied backwards. From the resulting sequence of rule applications it is apparent that the protocols error occurs because a passive process which forwarded the termination message can be activated afterwards. Hence the termination message and the process activation zone both move around the ring, but never meet.

▶ **Definition 15.** For a hypergraph G a *path* of length n is a sequence $v_0, e_1, v_1, \ldots, v_{n-1}, e_n, v_n$ of nodes $v_i \in V_G$ and edges $e_i \in E_G$, where e_i is adjacent to v_{i-1}, v_i and no node or edge occurs more than once. A n-bounded path graph is a hypergraph G in which all paths have length less than or equal to n.

The following proposition is inspired by results in [20].

▶ **Proposition 16.** *If it is known that every graph reachable from G_0 is an n-bounded path graph, then the coverability problem is decidable for the given GTS.*

Proof sketch. We prove well-structuredness of GTS wrt. subgraph ordering in the class of n-bounded path graphs. We can then apply the symbolic backward reachability algorithm in Section 3.1 by instantiating the subsumption relation $\sqsubseteq$ with subgraph inclusion, and by discarding all predecessors that do not satisfy the n-bounded path property. Termination is guaranteed here by the fact that subgraph ordering for bounded path graphs is a wqo [10]. ◀

3.6 Graph Transformation with Deletion by Minor Rules

The class of graph transformation systems with minor rules (node/edge deletion, edge contraction) is interesting in its own right. One can allow any combination of those rules and study decidability questions.

A direct consequence of the decidability of the coverability problem (wrt. the minor ordering) for graph transformation systems with edge contraction rules [17] is the decidability of the reachability problem for the class of graph transformation systems that contain

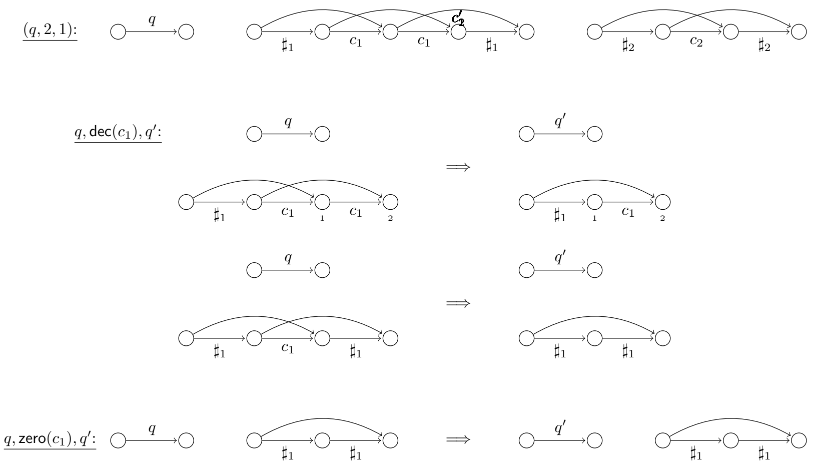


Figure 8 Encoding of a Minsky machine.

node deletion, edge contraction and edge deletion rules. In fact, for these kind of systems, reachability can be reduced to coverability because all the rules which are used in the minor ordering are also present in the system. Here we investigate what happens to the reachability problem when considering graph transformation systems where not all three minor rules are present. In the sequel, we will say that a graph transformation system is:

- *edge-contracting* if the set of rules contains all edge contraction rules for each edge label, excluding the edge deletion rule which is a special case of edge contraction;
- *edge-deleting* if the set of rules contains edge deletion rules for each edge label;
- *node-deleting* if the set of rules contains a node deletion rule.

We have the following result which shows that to obtain the decidability of the reachability problem in graph transformation systems with minor rules, all three types of minor rules are necessary.

▶ **Proposition 17.** *The reachability problem is undecidable for the following classes of graph transformation systems: (a) edge-deleting and node-deleting; (b) edge-contracting and node-deleting; (c) edge-deleting and edge-contracting.*

Proof sketch. We encode reachability for two counter machines. A two counter machine consists of a finite set of instructions manipulating two counters c_1 and c_2. Instructions have the following form (the semantics is the intuitive one): $(q, \mathsf{inc}(c_i), q')$ (increment c_i), $(q, \mathsf{dec}(c_i), q')$ (decrement c_i), $(q, \mathsf{zero}(c_i), q')$ (a blocking zero-test on c_i), where q, q' are control states. Given a (q, k, l) with $k, l \in \mathbb{N}$, it is undecidable to determine whether the program can reach the configuration (q', k', l') starting from $(q_0, 0, 0)$.

A configuration like $(q, 2, 1)$ is encoded as a path with additional crossing edges as shown Fig. 8. Such an encoding of configurations ensures that deletion by minor rules either blocks a simulation or introduces some garbage that cannot be removed. In both cases we define a reachability query that ensures that the simulation will not take wrong turns. Decrement and increment are encoded via graph transformations that preserve the structure of the representation as in Fig. 8. The correctness of the reduction is discussed in [3]. ◀

▶ **Proposition 18.** *The coverability problem is undecidable for graph transformation systems with edge deletion and node deletion rules.*

Proof sketch. The proof is a variant of the proof of Proposition 17 in the case of edge deletion and node deletion. More details can be found in [3]. ◀

3.7 Relabelling Rules

We now consider rules with arbitrary left-hand sides that are however only allowed to relabel their nodes or edges. So far, node labels were of no specific interest, but they play an important role in this section. Hence we now generalize the notion of hypergraph to include node labels: to a quadruple of the form $G = (V_G, E_G, c_G, l_G^E)$ representing a graph as introduced in Definition 1, we will now add an additional node-labelling function $l_G^V : V_G \to \Lambda'$, where Λ' is a finite set of node labels. The notion of graph morphism and the notion of rewriting are extended in the obvious way.

▶ **Definition 19.** A rewriting rule $r : L \rightharpoonup R$ is called a *relabelling rule* if r is a bijective (but not necessarily label-preserving) morphism. A node or edge x is said to be relabelled if $l_L(x) \neq l_R(r(x))$.

Example 4 can be seen as a relabelling system when deleting the lossy system rules as well as the rule creating new processes. The termination message and flag can be realised using two node labels or two edge labels where one indicates the existence and one the non-existence of the message and flag respectively.

Since reachability and coverability from a fixed initial graph are clearly decidable in this setting (the set of derivable graphs is finite), we consider the following *existential coverability* problem: assume that we are given a set of rewriting rules $\mathcal{R}$, a set of initial labels and a final graph G_f. Is there a graph G_0 labelled with only initial labels and a graph H such that $G_0 \Rightarrow_{\mathcal{R}}^* H$ and G_f is a subgraph of H?

The existential coverability is of interest when analysing distributed systems. By modelling an algorithm for a distributed system using a GTS, one effectively obtains a relabelling GTS because the system's topology remains unchanged during execution of the algorithm. When G_f represents an error configuration, the existential coverability problem transforms to the question: is there a distributed system where the modelled algorithm produces the specified error? A slightly different approach using node labels is pursued in [6], where the set of labels may be infinite and problems more specific for distributed systems are studied.

We first consider GTS with either only edge relabelling or only node relabelling rules, i.e. either the set of node labels or the set of edge labels is a singleton.

▶ **Proposition 20** (Edge Relabelling). *The existential coverability problem is decidable for graph transformation systems with relabelling rules where the set of node labels Λ' is a singleton, i.e., $|\Lambda'| = 1$.*

Proof sketch. Coverability can be decided by a simple fixed-point computation which determines the set of "reachable" edge labels. A label is reachable if it is initial or occurs on a right-hand side of some rule, where all labels of the corresponding left-hand side are reachable. Then the graph G_f is coverable if and only if its edges are labelled with only reachable labels.

Assume all labels of G_f are reachable. Then for every edge e_i of G_f there is an initial graph G_{e_i} and a sequence of relabelling steps leading to some graph covering the edge. By taking all graphs G_{e_i} and merging appropriate nodes, an initial graph can be obtained from which G_f is coverable by combining the relabelling steps used to cover the individual edges. The nodes must thereby be merged such that the covered edges are connected to each other in a proper way to form G_f after relabelling. It can also be shown that G_f is not coverable if it contains a non-reachable label. The full proof can be found in [3]. ◀

▶ **Proposition 21** (Node Relabelling). *The existential coverability problem is decidable for graph transformation systems with relabelling rules where the set of edge labels* Λ *is a singleton, i.e.,* $|\Lambda| = 1$.

Proof sketch. The proof for this proposition is analogous to the proof of Proposition 20 using node labels instead of edge labels. An initial graph for the whole graph can be constructed by taking one initial graph for each occurrence of a node label (covering that label) and adding every possible edge (i.e., generating a complete graph). ◀

Now assume there are both node and edge labels and both can be modified, then the existential coverability problem is undecidable. The graph structure cannot be disregarded in this case and therefore the main assumption of the proofs above does not hold.

▶ **Proposition 22** (Node and Edge Relabelling). *The existential coverability problem is undecidable for graph transformation systems with (general) node and edge relabelling rules.*

Proof sketch. A GTS can be constructed, which simulates a Turing machine. It first extracts a path out of the initial graph and then simulates a Turing machine with the path as tape, where each edge label is a cell of the tape. The node labels are thereby used to ensure that the extracted structure is actually a path, i.e. they ensure that at most two edges attached to the node belong to the tape. The Turing machine computation terminates if and only if there is a sufficiently large initial graph, containing a large enough tape and resulting in a graph covering the final state. The full proof can be found in [3]. ◀

Hence we arrive at the surprising conclusion that while for edge or node relabelling only their existential coverability is easily decidable, their combination results in an undecidable problem. The fact that graphs with edge and node labels can be encoded into graphs with edge labels does not help, since the encoding is not surjective and the corresponding relabelling problems cannot be reduced to each other: there could always be a graph outside of the image of the encoding from which we can cover the given subgraph.

4 Related Work

In this paper we focused our attention on single-pushout (SPO) rewriting. Another possibility would be to use double-pushout (DPO) rewriting [7]. In DPO a node cannot be deleted if it is connected to edges that are not explicitly deleted. The relation between Petri nets and DPO GTS has been studied in [2], where the encoding of nets into GTS deletes and recreates nodes in order to simulate the effects of inhibitor arcs from which we get undecidability of reachability and coverability even for rules that maintain the number of nodes always constant.

Well-structuredness of concurrency models in the class of bounded path graphs has been considered e.g. in [20, 23]. In all above mentioned models reduction rules have a restricted form to model either rendezvous or broadcast communication. In this paper we generalize well-structuredness to reduction rules defined by total injective morphisms. Well-structured graph rewriting is also considered in [9] where reduction rules that involve all neighbours (independently from their actual number) of a node are used to model broadcast communication. This type of rules cannot be modelled via GTS. Extending the language so as to capture broadcast communication is a possible future research direction.

Decidability boundaries for reachability problems of fragments of a graph-based model of biological systems are given in [8]. In κ a configuration consists of a graph in which nodes have labels and have a fixed number of binding sites. Rules can test and update node labels and the presence or absence of a binding site. Undecidability of coverability in GTS without deletion is inspired by a similar result for κ. The proof for GTS however does not require (to test on) node labels: it is uniquely based on an increasing graph structure used to simulate the evolution of an unbounded tape of a Turing machine. Furthermore, we consider here several other classes of reduction rules (e.g. context-free, minor and bounded path, relabeling) that are not studied in [8].

5 Conclusion

The results concerning decidability of reachability and coverability (excluding the relabelling cases) can be summarised in the diagram in Fig. 9. Interestingly, for some classes of GTS the reachability problem is decidable and the coverability problem undecidable, while for other classes it is the other way round. Of particular interest is the case of non-deleting GTS, into which we can encode Turing machines, however without guaranteeing termination for halting machines. Hence these GTS cannot be considered Turing-complete in the sense discussed in [19].

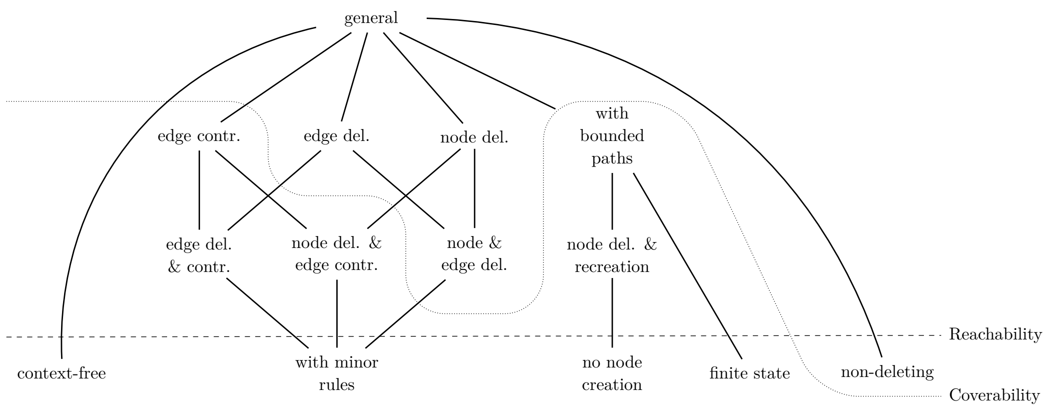


Figure 9 Decidability and Undecidability Boundaries.

We have obtained very general (un)decidability results that can be applied to all kinds of dynamic systems with evolving topologies. They can serve as a general toolbox to obtain decidability results for process calculi, which can often be straightforwardly encoded into graph transformation, for biological systems and for other formalisms. Note that, although we used hypergraphs, our undecidability proofs are formulated such that they also hold for directed multigraphs.

The studied subclasses of GTS can also be found in practice: first, in many examples in the literature the system is indeed fairly static and the number of nodes is fixed. Still, GTS here have a modelling advantage over Petri nets. Furthermore the node/edge deletion and edge contraction rules can be used to faithfully model lossy systems. Finally, triple graph grammars, specifying model transformations, usually have non-deleting rules.

Several of the decision procedures listed in this paper, especially those for coverability, can be implemented in practice and have reasonable runtimes. For instance, there are implementations of the coverability algorithm for Petri nets and of the backwards search algorithm described in [17].

One unsolved problem remains: the exact complexity of coverability for context-free GTS. We currently know that the problem is in PSPACE and we have an NP-hardness proof, but the exact complexity is open. We hope that this paper will stimulate further research in this area and lead to a better understanding of the algorithmic aspects of graph transformation systems.

Acknowledgements: We would like to thank Roland Meyer for interesting discussions about several aspects of this work.

References

1 P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proc. of LICS '96*, pages 313–321. IEEE, 1996.

2 P. Baldan, A. Corradini, and U. Montanari. Relating SPO and DPO graph rewriting with petri nets having read, inhibitor and reset arcs. In *Proc. of PNGT '04*, volume 127.2 of *ENTCS*, pages 5–28, 2005.

3 Nathalie Bertrand, Giorgio Delzanno, Barbara König, Arnaud Sangnier, and Jan Stückrath. On the decidability status of reachability and coverability in graph transformation systems. Technical Report DISI-TR-11-04, Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova, 2012.

4 F. Bonchi, F. Gadducci, and G. V. Monreale. Reactive systems, barbed semantics, and the mobile ambients. In *Proc. of FOSSACS '09*, pages 272–287. Springer, 2009. LNCS 5504.

5 N. Busi and G. Zavattaro. Deciding reachability in mobile ambients. In *Proc. of ESOP '05*, pages 248–262. Springer, 2005. LNCS 3444.

6 J. Chalopin, Y. Métivier, and W. Zielonka. Election, naming and cellular edge local computations. In *Proc. of ICGT '04 (International Conference on Graph Transformation)*, pages 242–256. Springer, 2004. LNCS 3256.

7 A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation—part I: Basic concepts and double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*, chapter 3. World Scientific, 1997.

8 G. Delzanno, C. Di Giusto, M. Gabbrielli, C. Laneve, and G. Zavattaro. The *kappa*-lattice: Decidability boundaries for qualitative analysis in biological languages. In *Proc. of CMSB '09*, pages 158–172. Springer, 2009. LNCS 5688.

9 G. Delzanno, A. Sangnier, and G. Zavattaro. Parameterized verification of ad hoc networks. In *Proc. of CONCUR '10*, pages 313–327. Springer, 2010. LNCS 6269.

10 G. Ding. Subgraphs and well-quasi-ordering. *Journal of Graph Theory*, 16(5):489–502, 1992.

11 F. Drewes, H.-J. Kreowski, and A. Habel. Hyperedge replacement graph grammars. In *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*, chapter 2. World Scientific, 1997.

12 C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. of ICALP '98*, pages 103–115. Springer, 1998. LNCS 1443.

13 H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. Algebraic approaches to graph transformation—part II: Single pushout approach and comparison with double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformation, Vol.1: Foundations*, chapter 4. World Scientific, 1997.

14 C. Ermel, H. Ehrig, F. Orejas, and G. Taentzer, editors. *Proc. of GraMoT '10*, volume 30 of *Electronic Communications of the EASST*, 2010.

15 J. Esparza and M. Nielsen. Decidability issues for Petri nets. Technical Report RS-94-8, BRICS, May 1994.

16 A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

17 S. Joshi and B. König. Applying the graph minor theorem to the verification of graph transformation systems. In *Proc. of CAV '08*, pages 214–226. Springer, 2008. LNCS 5123.

18 S. Joshi and B. König. Applying the graph minor theorem to the verification of graph transformation systems. Technical report, Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, 2012. Corrected version, available from `http://duepublico.uni-duisburg-essen.de/go/technische-berichte`.

19 S. Maffeis and I. Phillips. On the computational strength of pure ambient calculi. *Theoretical Computer Science*, 330:501–551, 2005.

20 R. Meyer. *Structural Stationarity in the π-Calculus.* PhD thesis, Carl-von-Ossietzky-Universität Oldenburg, 2009.

21 A. Rensink. The GROOVE simulator: A tool for state space generation. In *Proc. of AGTIVE '03*, pages 479–485. Springer, 2003. LNCS 3062.

22 Neil Robertson and Paul Seymour. Graph minors XXIII. Nash-Williams' immersion conjecture. *Journal of Combinatorial Theory Series B*, 100:181–205, March 2010.

23 T. Wies, D. Zufferey, and T. A. Henzinger. Forward analysis of depth-bounded processes. In *Proc. of FOSSACS '10*, pages 94–108. Springer, 2010. LNCS 6014.

Normalisation for Dynamic Pattern Calculi

Eduardo Bonelli[1,2], Delia Kesner[3], Carlos Lombardi[1,3,4], and Alejandro Ríos[4]

1 Univ. Nac. de Quilmes, Argentina
2 CONICET, Argentina
3 Univ. Paris Diderot, Sorbonne Paris Cité, PPS, CNRS, France
4 Univ. de Buenos Aires, Argentina

Abstract

The Pure Pattern Calculus (PPC) [10, 11] extends the λ-calculus, as well as the family of algebraic pattern calculi [20, 6, 12], with first-class patterns *i.e.* patterns can be passed as arguments, evaluated and returned as results. The notion of *matching failure* of PPC in [11] not only provides a mechanism to define functions by pattern matching on cases but also supplies PPC with parallel-or-like, non-sequential behaviour. Therefore, devising normalising strategies for PPC to obtain well-behaved implementations turns out to be challenging.

This paper focuses on normalising reduction strategies for PPC. We define a (multistep) strategy and show that it is normalising. The strategy generalises the leftmost-outermost strategy for λ-calculus and is strictly finer than parallel-outermost. The normalisation proof is based on the notion of *necessary set of redexes*, a generalisation of the notion of needed redex encompassing non-sequential reduction systems.

1998 ACM Subject Classification F.4.1 Mathematical Logic, D.3.3 Language Constructs and Features

Keywords and phrases Pattern calculi, reduction strategies, sequentiality, neededness

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.117

Category Regular Research Paper

1 Introduction

Pattern calculi [20, 12, 6, 15] model the pattern-matching primitives of functional programming languages (*e.g.* OCAML, ML, Haskell) and proof assistants (*e.g.* Coq, Isabelle), where functions can be defined by n different *cases* by writing for example $\texttt{f} := p_1\texttt{->}s_1 \mid \ldots \mid p_n\texttt{->}s_n$. When p_i is typically expressed in terms of *constructors* and variables we speak of *algebraic pattern calculi*. The application of $\texttt{f}$ to an argument u starts by matching p_1, the *pattern* of the first case, against u. If such a matching is successful, then it yields a substitution σ_1 whose domain is the set of variables in p_1. This substitution is applied to s_1, the *body* of the first case, and then the substituted body is evaluated. If a successful matching is not possible, *i.e.* there is a *matching failure*, then evaluation continues with the following cases in the definition of $\texttt{f}$.

The Pure Pattern Calculus (PPC). PPC [10, 11] generalises algebraic pattern calculi (and hence the λ-calculus) by allowing an *arbitrary* term t_i to take the role of a pattern p_i thus generalising a function to $\texttt{f} := t_1\texttt{->}s_1 \mid \ldots \mid t_n\texttt{->}s_n$. Reduction inside the t_i is now allowed, hence patterns may be computed *dynamically*. Also, symbols in t_i now play two roles: either they are *variables* (*i.e.* place holders) for terms (which substitution from "outside" will duly fill in) or they are *matchables* (depicted with a hat for easy identification) in the sense that they are only used to decompose data. These two roles are distinguished by decorating a case t`->`s with a set θ of *binding symbols* that singles out the subset of matchables in the pattern t that are meant for matching, the variables in the body s being those acting as place-holders. Consider for example $\texttt{elim} ::= \lambda_{\{x\}}\ \widehat{x}.(\lambda_{\{y\}}\ x\ \widehat{y}.y)$ where we write $\lambda_\theta\ t.s$ rather than $\texttt{t} \ \texttt{->}_\theta\ \texttt{s}$. The inner abstraction $\lambda_{\{y\}}\ x\ \widehat{y}.y$ binds the only occurrence of the *matchable* $\widehat{y}$ in the pattern $x\ \widehat{y}$ and that of the *variable* y in the body y; the x in $x\ \widehat{y}$ is excluded from $\{y\}$ since it acts as a place-holder in that pattern. However, the occurrence of x, as well as that of $\widehat{x}$, are both bound by the outermost $\lambda_{\{x\}}$, as can be seen in Fig. 1.

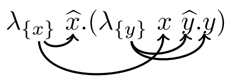


Figure 1

The dynamics of reduction in `PPC` may be illustrated by the reduction sequence of Fig. 2 in which `elim` is applied to the function $\lambda_{\{z\}}\ \widehat{z}.\texttt{cons}\ z\ \texttt{nil}$, where `cons` and `nil` denote constructors. In the first step, $\lambda_{\{z\}}\ \widehat{z}.\texttt{cons}\ z\ \texttt{nil}$ is substituted for x into the pattern $x\ \widehat{y}$. In the second step, the resulting application $(\lambda_{\{z\}}\ \widehat{z}.\texttt{cons}\ z\ \texttt{nil})\,\widehat{y}$, which resides in a pattern, is reduced. The resulting term, when applied to an argument, will yield a succesful matching only if this argument is a *compound data* of the form $\texttt{cons}\ t\ \texttt{nil}$. This example exhibits the *pattern polymorphism* capabilities of `PPC` allowing to obtain structurally different deconstructors by applying the same term `elim` to different arguments.

$$
\begin{aligned}
&(\lambda_{\{x\}}\ \widehat{x}.(\lambda_{\{y\}}\ x\ \widehat{y}.y))\,(\lambda_{\{z\}}\ \widehat{z}.\texttt{cons}\ z\ \texttt{nil})\\
\to\ &\lambda_{\{y\}}\ (\lambda_{\{z\}}\ \widehat{z}.\texttt{cons}\ z\ \texttt{nil})\,\widehat{y}.y\\
\to\ &\lambda_{\{y\}}\ \texttt{cons}\ \widehat{y}\ \texttt{nil}.y
\end{aligned}
$$

Figure 2

Reduction in Fig. 2 proceeded smoothly. However, it may fail. In `PPC`, $(\lambda_\theta\ p.s)t$ is reducible only if the match of the pattern p against the argument t is *decided*, *i.e.* it is either a successful match, specified by a substitution, or a matching failure, specified by a closed *normal form* (written[1] ***nf***). *E.g.* consider $(\lambda_{\{x\}}\ \texttt{a}\ r_1.x)(\texttt{b}\ r_2)$ where r_1 and r_2 are *redexes*. Matching of $\texttt{a}\ r_1$ against $\texttt{b}\ r_2$ fails since the head constructors `a` and `b` are different. This is also the case for $(\lambda_{\{x\}}\ \texttt{a}\ r_1\ \texttt{b}.x)(\texttt{a}\ r_2\ \texttt{d})$, now because the constructors `b` and `d` mismatch, a fact that can be established without examining r_1 or r_2. Indeed, failure of matching for *any* component of a given compound data automatically implies failure of matching for the entire compound.

Non-sequentiality of `PPC`. What would be a smart *normalising* strategy for `PPC`? Clearly, if the matching of p against t is decided when evaluating $(\lambda_\theta p.s)t$, then it is a redex and should be selected. However, when it is non-decided, it is necessary to understand which subterm (p, s or t) needs to be evaluated first in order to attain a normal form for the whole term. *E.g.* it is the *pattern* that must be reduced in $(\lambda_{\{x\}}\ \texttt{I}\ (\texttt{a}\ \widehat{x}).x)(\texttt{a}\ \texttt{c})$ (`I` is the identity function) in order to attain a decided match, the *argument* in $(\lambda_{\{x\}}\ \texttt{a}\ \widehat{x}.x)(\texttt{I}\ (\texttt{a}\ \texttt{c}))$, and *both* the pattern and the argument in $(\lambda_{\{x\}}\ \texttt{a}\ (\texttt{I}\ (\texttt{b}\ \widehat{x})).x)((\texttt{I}\ \texttt{a})\ \texttt{b}\ y)$. Even though we could establish that the pattern or the argument or both have to be reduced, deciding *which* redex to pick in each of these cases is not always possible. An example is $t_1 := (\lambda_{\{x\}}\ \texttt{a}\ (\texttt{b}\ \widehat{x})\ r_1.r_2)\ (\texttt{a}\ r_3\ (\texttt{d}\ r_4))$ where matching is non-decided: both pattern and argument must be reduced in order for t_1 to become a redex. Reducing r_1 is not necessarily a good idea. Indeed, suppose r_1 is any non-terminating computation and r_3 reduces to $\texttt{d}\ t$, for some t. Then reduction of r_3 causes

[1] There are other different reasonable approaches.

a matching failure since `b` and `d` mismatch. We could rather have selected r_3, however if this time it is r_3 that is non-terminating and r_1 reduces to `b` t, for some t, then again we miss failure and loop. The same happens if we choose r_4. The term t_1 illustrates that `PPC` fails to be *sequential*.

Informally, sequentiality means that given a term of the form $C[r_1, \dots, r_n]$ where C does not contain any redex and every r_i is a redex, there exists an index i s.t. r_i is a *needed* redex and the choice of i is independent from $r_1, \dots, r_n$. A redex r in a term t is needed iff every reduction sequence from t to normal form reduces (a *residual* of) r. Another example showing terms not in normal form may not have needed redexes is $t_2 :=$ $(\lambda_{\{y\}}$ `a b c` $\widehat{y}.y)$ (`a` (`I c`) (`I b`) (`I a`)) (redexes are shown in gray). This term admits at least two different reduction sequences to normal form: $t_2 \to (\lambda_{\{y\}}$ `a b c` $\widehat{y}.y)$(`a c` (`I b`) (`I a`)) $\to$ ***nf*** and also $t_2 \to (\lambda_{\{y\}}$ `a b c` $\widehat{y}.y)$(`a` (`I c`) `b` (`I a`)) $\to$ ***nf***. Sequentiality fails in `PPC` because matching may fail for *different reasons*: none of the redexes in t_2 is needed since failure of matching can be declared in terms of (only) `I b` or `I c`.

Even if dynamic patterns and non-sequentiality are central issues of this paper, the calculus would also be non-sequential if the set of patterns were restricted to the static ones. As explained before, non-sequentiality comes from the notion of matching failure introduced in [11], which is detailed in Sec. 2, and applies to any kind of static pattern, including the standard, algebraic ones.

Towards normalisation strategies for `PPC`. It has been shown in different settings that repeated contraction of needed redexes yields normalising strategies for sequential rewriting systems. Failure of sequentiality leaves us with two avenues to pursue in order to devise normalising strategies for `PPC`. The first is to introduce *look-ahead* by performing several reduction steps in order to identify an i. In this case, apart from C we would have to examine the r_is. Such an approach is overly expensive since it involves testing for cycles [1]. The second avenue consists in selecting a *multistep*: a *set* of redexes reduced simultaneously at each step. Of particular appeal in this approach is the notion of *necessary set of redexes*: a set of redexes $\mathcal{A}$ in a term t is called necessary iff every reduction sequence from t to normal form reduces at least (a *residual* of) one element of $\mathcal{A}$, even if this element may differ from sequence to sequence. The set of all redexes, and also the set of all outermost redexes, are necessary sets for any term in `PPC`. However, implementations could benefit from *finer* strategies, *i.e.* reduction strategies selecting smaller sets of redexes in each multistep.

Contributions. We propose a strategy for `PPC` which selects for each term a necessary set bounded by the set of its outermost redexes and show that it is normalising. More precisely, we introduce:

- a theory of needed normalisation for `PPC` by adapting the notion of *necessary sets* [21] to the higher-order case;
- a proof that repeated contraction of *necessary sets* of redexes is normalising, provided that those sets also enjoy an additional property, namely they are *non-gripping* [16];
- an inductively formulated strategy for `PPC` that produces necessary, non-gripping sets of redexes, and hence is normalising.

Related work. In λ-calculus and, more generally, orthogonal higher-order (HO) Expression Reduction Systems, any normalising term not in normal form contains a needed redex and (one-step) contraction of needed redexes attains a normal form [5, 7]. Unfortunately, as discussed above, terms in `PPC` not in normal form may contain no needed redexes. Multistep reduction strategies have been studied for both first-order (FO) and HO rewriting. Normalising multistep strategies by means of necessary sets have been defined for non-sequential FO TRS in [21], which has been our main source of inspiration. Van Raamsdonk [23] extends O'Donnell's result [19] (reduction of all outermost redexes is normalising) to almost orthogonal HORS. Indeed, [23] proves that being *outermost fair* is a sufficient condition for a reduction strategy to be normalising; this result was then extended [22] to weakly orthogonal HO rewriting. These works have influenced some of the concepts and technical tools used in this paper. Melliès [16, 17] develops an axiomatic theory of neededness for (possibly) overlapping rewrite systems, motivated by the work of Huet and Lévy for orthogonal FO TRS [8]. Pattern calculi may be modeled as the axiomatic rewrite systems of Melliès, however his normalisation results are not applicable since pattern calculi do not enjoy *stability* [16](Axiom IV, pg.80): there may be more than one way to create the same redex due to matching failure.

Regarding the specific setting of non-sequential pattern calculi (including matching failure) we are not aware of any literature on normalising strategies. For an abridged version of `PPC` resulting from restricting the set of patterns to the algebraic ones and from disallowing matching to fail, one-step *standard* reductions (obtained by means of a standardisation theorem) turn out to be normalising [13]. Also worth mentioning is the existence of sequential (*i.e.* every term has a needed redex) operational semantics of dynamic pattern calculi appearing for example in [9, 3, 4]; they can be understood as (re)formulations of `PPC` where the conditions determining when a match should fail impose a more restricted evaluation order.

Structure of the paper. Sec. 2 introduces `PPC`. Sec. 3 defines multistep strategies and develops the tools needed to guarantee that complete developments can be used as multisteps. Sec. 4 presents a reduction strategy $\mathcal{S}$ for `PPC`. Sec. 5 formalises the notion of necessary sets of redexes, motivates and introduces the additional non-gripping property, shows that $\mathcal{S}$ always reduces necessary and non-gripping sets, and uses these facts to prove that it is normalising. Finally, Sec. 6 concludes and suggests further work.

2 The Pure Pattern Calculus

This section briefly introduces `PPC` following the presentation of [11][2].

Syntax: Consider a countable set of **symbols** $f, g, \ldots, x, y, z$. Sets of symbols are denoted by meta-variables θ, ϕ, ... The syntax of `PPC` is summarised by the following grammar:

$$\begin{array}{llcll}
\textbf{Terms} & (\boldsymbol{T}) & t & ::= & x \mid \widehat{x} \mid tt \mid \lambda_\theta\ t.t \\
\textbf{Data-Structures} & (\boldsymbol{DS}) & D & ::= & \widehat{x} \mid Dt \\
\textbf{Matchable-forms} & (\boldsymbol{MF}) & F & ::= & D \mid \lambda_\theta\ t.t
\end{array}$$

The term x is called a **variable**, $\widehat{x}$ a **matchable**, tu an **application** (t is the **function** and u the **argument**) and $\lambda_\theta\ p.u$ an **abstraction** (θ is the set of **binding symbols**, p is the **pattern** and u is the **body**). Application (resp. abstraction) is left (resp. right) associative. A λ-abstraction $\lambda x.t$ can be defined by $\lambda_{\{x\}}\ \widehat{x}.t$. The **identity function** $\lambda_{\{x\}}\ \widehat{x}.x$ is abbreviated `I`.

[2] Other presentations are [10, 9], but both of them use sequential, rather than parallel-or like, semantics.

A binding symbol $x \in \theta$ of an abstraction $\lambda_\theta\ p.s$ *binds* matchable occurrences of x in p and variable occurrences of x in s. The derived notions of **free variables** and **free matchables** are respectively denoted by $\texttt{fv}(\_)$ and $\texttt{fm}(\_)$. This is illustrated in Fig. 3. As usual, we consider terms up to **alpha-conversion**, *i.e.* up to renaming of bound matchables and variables. **Constructors** are matchables which are not bound and, to ease the presentation, they are often denoted in typewriter fonts $\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \ldots$, thus for example $\lambda_{\{x,y\}}\ \widehat{x}\ y\ \texttt{a}.y$ denotes $\lambda_{\{x,y\}}\ \widehat{x}\ y\ \widehat{z}.y$. The distinction between matchables and variables is unnecessary for standard (static) patterns which do not contain free variables.

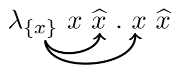


Figure 3

A **position** is either ϵ (the empty position), or na, where $n \in \{1,2\}$ and a is a position. We use $a, b, \ldots$ (resp. $\mathcal{A}, \mathcal{B}, \ldots$ and $\delta, \rho, \pi, \ldots$) to denote positions (resp. sets and sequences of positions) and $b\mathcal{A}$ to mean $\{ba \mid a \in \mathcal{A}\}$. The **set** $\texttt{Pos}(t)$ **of positions** of t is defined as expected, provided that for abstractions $\lambda_\theta\ p.s$ positions inside both p and s are considered. Here is an example $\texttt{Pos}(\lambda_{\{x\}}\ \texttt{a}\ \texttt{b}.\texttt{a}\ x\ x) = \{\epsilon, 1, 2, 11, 12, 21, 22, 211, 212\}$.

We write $t|_a$ for the **subterm of** t **at position** a and $t[s]_a$ for the **replacement** of the subterm at position a in t by s. Notice that replacement may capture variables. We write $a \leq b$ (resp. $a \parallel b$) when the position a is a **prefix** of (resp. **disjoint** from) the position b. All these notions are defined as expected [2] and extended to sets of positions as well.

Substitution and Matching: A **substitution** σ is a mapping from variables to terms with finite domain $\texttt{dom}(\sigma)$. A **match** μ is either a substitution or a special constant in the set $\{\texttt{fail}, \texttt{wait}\}$. A **match** is **positive** if it is a substitution; it is **decided** if it is either positive or $\texttt{fail}$. The notions of domain and free variables/matchables are naturally extended to matches, in particular, the domain of $\texttt{fail}$ is the empty set and that of $\texttt{wait}$ is undefined. The **application of a substitution** σ to a term is written and defined as usual on alpha-equivalence classes. The **application of a match** μ to a term t, written μt, is defined as follows: if μ is a substitution, then it is applied as explained above; if $\mu = \texttt{wait}$, then μt is undefined; if $\mu = \texttt{fail}$, then μt is the identity function $\texttt{I}$. Other *closed terms in normal form* could be taken to define the last case, this one allows in particular to encode pattern-matching definitions given by alternatives [11].

The **disjoint union** of two matches μ_1 and μ_2 is a crucial operation used to define the operational semantics of $\texttt{PPC}$. Disjoint union is written $\mu_1 \uplus \mu_2$ and is defined as: their union if both μ_i are substitutions and $\texttt{dom}(\mu_1) \cap \texttt{dom}(\mu_2) = \emptyset$; $\texttt{wait}$ if either of the μ_i is $\texttt{wait}$ and none is $\texttt{fail}$; $\texttt{fail}$ otherwise. This definition of disjoint union of matches validates the following equations which are responsible for the non-sequential nature of $\texttt{PPC}$:

$$\texttt{fail} \uplus \texttt{wait} = \texttt{wait} \uplus \texttt{fail} = \texttt{fail}$$

We return to these equations immediately after the definition of the operational semantics of the calculus. The **compound matching operation** takes a term, a set of binding symbols and a pattern and returns a match, it is defined by applying the following equations in order:

$$\begin{array}{rcll}
\{\!\{\widehat{x} \rhd_\theta t\}\!\} & := & \{x \to t\} & \text{if } x \in \theta \\
\{\!\{\widehat{x} \rhd_\theta \widehat{x}\}\!\} & := & \{\} & \text{if } x \notin \theta \\
\{\!\{pq \rhd_\theta tu\}\!\} & := & \{\!\{p \rhd_\theta t\}\!\} \uplus \{\!\{q \rhd_\theta u\}\!\} & \text{if } tu, pq \in \boldsymbol{MF} \\
\{\!\{p \rhd_\theta t\}\!\} & := & \texttt{fail} & \text{if } p, t \in \boldsymbol{MF} \\
\{\!\{p \rhd_\theta t\}\!\} & := & \texttt{wait} & \text{otherwise}
\end{array}$$

The use of disjoint union in the third case of the previous definition restricts compound matching to linear patterns [3], which is necessary to guarantee confluence. Indeed, disjoint union of two substitutions fails whenever their domains are not disjoint.

The result of the **matching operation**[4] $\{p/_\theta\ t\}$ is defined to be the *check* of $\{\!\{p \rhd_\theta t\}\!\}$ on θ; where the **check** of a match μ on θ is `fail` if μ is a substitution whose domain is not θ, μ otherwise. Notice that $\{p/_\theta\ t\}$ is never positive if p is not linear with respect to θ. We now give some examples: $\{\widehat{x}\widehat{x}/_{\{x\}}\ uv\}$ gives `fail` because $\widehat{x}\widehat{x}$ is not linear; $\{\widehat{x}\widehat{y}/_{\{x,y,z\}}\ uv\}$ gives `fail` because $\{x,y,z\} \neq \{x,y\}$, $\{\widehat{x}/_\emptyset\ u\}$ gives `fail` because $\emptyset \neq \{x\}$; $\{\widehat{y}/_{\{x\}}\ \widehat{y}\}$ gives `fail` because $\{x\} \neq \emptyset$; $\{\widehat{x}\widehat{y}/_{\{x\}}\ u\widehat{z}\}$ gives `fail` because $\{\!\{\widehat{y} \rhd_{\{x\}} \widehat{z}\}\!\}$ is `fail`; $\{\widehat{x}\widehat{y}/_\emptyset\ u\widehat{z}\}$ gives `fail` for the same reason.

Semantics: The semantics of `PPC` is given by means of the following **reduction rule**:

$$(\lambda_\theta\ p.s)u \mapsto \{p/_\theta\ u\}s, \text{ if } \{p/_\theta\ u\} \text{ is decided}$$

The rule applies to the term $(\lambda_{\{x,y\}}\ \mathtt{a}\ \widehat{x}\ \widehat{y}.y\ x)(\mathtt{a}\ \mathtt{b}\ (\mathtt{I}\ \mathtt{a}))$ yielding $(\mathtt{I}\ \mathtt{a})\ \mathtt{b}$; and also to $(\lambda_{\{x,y\}}\ \mathtt{a}\ \widehat{x}\ \widehat{y}.y\ x)(\mathtt{c}\ \mathtt{b}\ (\mathtt{I}\ \mathtt{a}))$ yielding `I` (since the matching operation yields `fail`). However, the rule does not apply to the term $(\lambda_{\{x,y\}}\ \mathtt{a}\ \widehat{x}\ \widehat{y}.y\ x)(\mathtt{I}\ \widehat{x})$ since $\{\mathtt{a}\ \widehat{x}\ \widehat{y}/_{\{x,y\}}\ \mathtt{I}\ \widehat{x}\}$ is `wait`.

It is worth noticing that sequentiality of `PPC` can be recovered (see *e.g.* [9, 3, 4]) by modifying the equations of disjoint union, however, some meaningful terms will no longer be normalising. Thus for example, if $\mathtt{fail} \uplus \mu$ is defined to be `fail`, while $\mathtt{wait} \uplus \mathtt{fail} = \mathtt{wait}$ and $\sigma \uplus \mathtt{fail} = \mathtt{fail}$, then $(\lambda_\emptyset\ \mathtt{a}\ \mathtt{b}\ \mathtt{b}\ .\widehat{y})(\mathtt{a}\ \Omega\ \mathtt{c})$, where Ω is a non-terminating term, would never fail as expected.

A (β) **redex** is a term of the form $(\lambda_\theta\ p.s)u$ s.t. $\{p/_\theta\ u\}$ is decided. A redex $(\lambda_\theta\ p.s)u$ s.t. $\{p/_\theta\ u\} = \mathtt{fail}$ is called a **matching failure**. A position $a \in \mathtt{Pos}(t)$ is called a **redex occurrence** of t iff $t|_a$ is a redex; $\mathcal{RO}(t)$ denotes the **set of all redex occurrences** of a term t. A redex in t is **outermost** if it is not contained in any other redex; *i.e.* it is minimal with respect to the order $<$ on redex occurrences. A **reduction step from t to s via a** is a tuple $t \xrightarrow{a} s$, where $t|_a$ is a redex $(\lambda_\theta\ p.u)v$ and $s = t[\{p/_\theta\ v\}u]_a$; this reduction step denotes the *contraction* (or *evaluation*) of the redex occurrence $a \in \mathcal{RO}(t)$. We occasionally identify a reduction step $t \xrightarrow{a} s$ with the redex occurrence a if no confusion arises.

Given a sequence of positions $\delta = a_1; \ldots; a_n; \ldots$ (possibly empty or infinite) and $t_0 \in \boldsymbol{T}$, a **reduction sequence from t_0 via δ**, written $t_0 \overset{\delta}{\twoheadrightarrow}$, is a sequence of the form $t_0 \xrightarrow{a_1} t_1 \ldots t_{n-1} \xrightarrow{a_n} t_n \ldots$. We write `nil` for the empty reduction sequence. We occasionally identify $t \overset{\delta}{\twoheadrightarrow} s$ with δ if no confusion arises. The term t **reduces to s in many steps**, written $t \twoheadrightarrow s$, iff there is a reduction sequence $t \overset{\delta}{\twoheadrightarrow} s$. Notice that $\twoheadrightarrow$ is the reflexive and transitive closure of $\rightarrow$. Given $t_0 \overset{\delta}{\twoheadrightarrow}$, where $\delta = a_1; \ldots; a_n; \ldots$, we write $\delta[i..k]$ $(1 \leq i \leq k \leq n)$ to identify the finite (sub)reduction sequence $t_{i-1} \xrightarrow{a_i} t_i \ldots t_{k-1} \xrightarrow{a_k} t_k$ and $\delta[i]$ $(1 \leq i \leq n)$ to identify the reduction step $t_{i-1} \xrightarrow{a_i} t_i$.

A term s is in **normal form**, written $s \in \boldsymbol{NF}$, iff there is no t s.t. $s \rightarrow t$. A term s is **normalising** iff there is a normal form t s.t. $s \twoheadrightarrow t$.

We refer the interested reader to [11] where different `PPC` examples are introduced, particularly to illustrate path and pattern polymorphism.

[3] A pattern p is linear w.r.t. θ if for every x in θ, the matchable $\widehat{x}$ appears at most once in p.

[4] Note that the notation for (compound) matching we have just given differs from [10] and [11]: the pattern and argument appear in reversed order there.

3 Multisteps and multireductions

In the light of the discussion in the Introduction on the inexistence of needed redexes, the reduction strategy for `PPC` we shall propose in Sec. 4 will select a *set* of redexes to contract at each step. Contraction of a set of redexes is understood as *simultaneous* contraction of all its members. Since, in principle, the order in which these members are contracted could affect the target term of the step, it becomes necessary to lay out precise definitions of what it means to perform simultaneous contraction of a set of redexes. It should be mentioned that these definitions are rather straightforward in first-order rewriting since the aforementioned set of redexes may be assumed to contain *pairwise disjoint* redexes, without any loss of generality. This owes to the fact that (first-order) residuals of such sets are again pairwise disjoint. In the higher-order case, however, this no longer holds, as can be seen by means of the following example. Consider the reduction step

$$t = (\lambda_{\{x\}}\ \widehat{x}.(\lambda_{\{y\}}\ \widehat{y}.y\ x)s)r \to (\lambda_{\{y\}}\ \widehat{y}.y\ r)s = u$$

where r is a redex and s an arbitrary term; the redexes $(\lambda_{\{y\}}\ \widehat{y}.y\ x)s$ and r are disjoint in t, while $(\lambda_{\{y\}}\ \widehat{y}.y\ r)s$ and r, their respective residuals in u (according to the formal definition given below), are not. This significantly complicates any effort of adapting extant results on normalisation of first-order systems to the higher-order setting.

Given a term t, $b \in \texttt{Pos}(t)$ and $a \in \mathcal{RO}(t)$, the **descendants of** b **after** a **in** t, written $b/^t a$ or simply b/a if the term is clear from the context, is the set of *positions* defined as follows:

$$\begin{array}{rl} \emptyset & \text{if}\ \ a = b. \\ \{b\} & \text{if}\ \ a \not\leq b. \\ \{an\} & \text{if}\ \ b = a12n, t|_a = (\lambda_\theta\ p.s)u,\ \text{and}\ \{p/_\theta\ u\}\ \text{is a substitution} \\ \{akn\ .\ s|_k = x\} & \text{if}\ \ b = a2mn, t|_a = (\lambda_\theta\ p.s)u, \{p/_\theta\ u\}\ \text{is a substitution}, p|_m = \widehat{x}\ \text{and}\ x \in \theta \\ \emptyset & \text{otherwise} \end{array}$$

If b is the position of a redex in t, then each position in b/a denotes a **residual** of b after performing a. This notion is extended to sets $\mathcal{B} \subseteq \mathcal{RO}(t)$ as follows: the **residuals of** $\mathcal{B}$ **after** a **in** t are $\mathcal{B}/a := \bigcup_{b\in\mathcal{B}} b/a$. In particular $\emptyset/a = \emptyset$. Given $t \overset{\delta}{\twoheadrightarrow} u$ and $\mathcal{B} \subseteq \mathcal{RO}(t)$, the **residuals of** $\mathcal{B}$ **after the sequence** δ, are: if $\delta = \texttt{nil}$, then $\mathcal{B}/\delta := \mathcal{B}$; otherwise $\mathcal{B}/\delta := (\mathcal{B}/\delta[1])/\delta[2\dots n]$ where $\delta = \delta[1..n]$.

Let $\mathcal{A} \subseteq \mathcal{RO}(t)$. The reduction sequence $t \overset{\delta}{\twoheadrightarrow}$ is a **development** of $\mathcal{A}$ iff $\delta[i] \in \mathcal{A}/\delta[1..i-1]$ for all i. A development δ of $\mathcal{A}$ is said to be **complete** iff δ is finite and $\mathcal{A}/\delta = \emptyset$.

Using standard techniques we show that the order in which contraction of redexes in a given set is performed does not introduce non-termination nor does it affect the target term.

▶ **Proposition 1** (Strong Finite Developments). *Let t be a term and $\mathcal{A} \subseteq \mathcal{RO}(t)$.*

(i) All developments of $\mathcal{A}$ from t are finite.

(ii) If $t \overset{\delta_i}{\twoheadrightarrow} u_i$ $(i = 1, 2)$ are two complete developments of $\mathcal{A}$, then $u_1 = u_2$ and $\forall a \in \mathcal{RO}(t)$, $a/\delta_1 = a/\delta_2$.

As a consequence, for every $\mathcal{A} \subseteq \mathcal{RO}(t)$ we can now formally define a **multistep** $t \overset{\mathcal{A}}{\multimap} u$ iff there is a complete development of $\mathcal{A}$ from t to u. Given a sequence of sets $\Delta = \mathcal{A}_1; \dots; \mathcal{A}_n; \dots$ (possibly empty or infinite) and $t_o \in \boldsymbol{T}$, a **multireduction sequence** Δ from t_0, written $t_0 \overset{\Delta}{\multimap\!\!\!\rightarrow}$, is a sequence of the form $t_0 \overset{\mathcal{A}_1}{\multimap} t_1 \dots t_{n-1} \overset{\mathcal{A}_n}{\multimap} t_n \dots$. We write `nil` for the empty multireduction sequence. We occasionally identify $t \overset{\Delta}{\multimap\!\!\!\rightarrow} s$ with Δ if no confusion arises. We use the notations $\Delta[i]$ and $\Delta[i..j]$ to denote $\mathcal{A}_i$ and the (sub)sequence $\mathcal{A}_i; \ldots ; \mathcal{A}_j$ respectively. We use $\Gamma, \Delta, \Pi, \Psi, \ldots$ to denote multireduction sequences. Let $\mathcal{A}$, $\mathcal{B}$ be sets of positions. The **residual of $\mathcal{B}$ after $\mathcal{A}$**, written $\mathcal{B}/\mathcal{A}$, is defined as the multistep $\mathcal{B}/\delta$ where δ is *any* complete development of $\mathcal{A}$ (this is well-defined by Prop. 1).

We extend the concept of residual to multireductions as well. The **residual of $\mathcal{B}$ after Δ** is defined as follows: if $\Delta = \texttt{nil}$, then $\mathcal{B}/\Delta := \mathcal{B}$; otherwise $\mathcal{B}/\Delta := (\mathcal{B}/\Delta[1])/\Delta[2..n]$ where $\Delta = \Delta[1..n]$. Analogously, we define the **residual of Δ after $\mathcal{B}$** as follows: $\texttt{nil}/\mathcal{B} := \texttt{nil}$, $\Delta[1..n]/\mathcal{B} := (\Delta[1]/\mathcal{B})/(\Delta[2..n]/(\mathcal{B}/\Delta[1]))$. By applying several times Prop. 1 it can be proved that $\Delta; (\mathcal{B}/\Delta)$ and $\mathcal{B}; (\Delta/\mathcal{B})$ end in the same term and induce the same residual relation.

Prop. 1 also allows us to introduce the **depth** of $\mathcal{A}$, written $\nu(\mathcal{A})$, a notion we shall use in Sec. 5. It is defined as the length of the longest complete development of $\mathcal{A}$. Since $\texttt{PPC}$ is finitely branching, this is well-defined by König's Lemma. We occasionally use the notation $\nu(\mathcal{A}, t)$ to make explicit the source of the multistep $\mathcal{A}$.

4 The reduction strategy

The rationale behind the reduction strategy for $\texttt{PPC}$ is that rather than selecting the entire set of outermost redexes of a given term t, this set is *refined* in two complementary ways. Let us call **preredex** a term of the form $(\lambda_\theta\, p.t)u$, regardless of whether the match $\{p/_\theta\, u\}$ is decided or not. The first observation about the strategy is that it focuses on the leftmost-outermost (LO) preredex of t, entailing that when $\texttt{PPC}$ is restricted to the λ-calculus it behaves exactly as the LO strategy for the λ-calculus. Second, if the match corresponding to the LO occurrence of a preredex is not decided, then the strategy selects only the (outermost) redexes in that subterm which should be contracted to get it "closer" to a decided match.

Suppose $(\lambda_\theta\, p.t)\, u$ is this LO preredex. If $\{p/_\theta\, u\}$ is decided, then the preredex (in fact a redex), is the only one selected by the strategy (it is LO in this case). If the match $\{p/_\theta\, u\}$ is not decided, then the strategy selects the outermost redexes whose contraction may *contribute* towards obtaining a decided match. More precisely, in the term $(\lambda_{\{x,y\}}\, \texttt{a}\, \widehat{x}\, (\texttt{c}\, \widehat{y}).y\, x)\, (\texttt{a}\, r_1\, r_2)$ the match $\{\texttt{a}\, \widehat{x}\, (\texttt{c}\, \widehat{y})/_{\{x,y\}}\, \texttt{a}\, r_1\, r_2\}$ is not decided and the role played by r_1 is different from that of r_2 in obtaining a decided match. Replacing r_1 by an arbitrary term t_1 will not yield a decided match, *i.e.* $\{\texttt{a}\, \widehat{x}\, (\texttt{c}\, \widehat{y})/_{\{x,y\}}\, \texttt{a}\, t_1\, r_2\}$ is not decided. However, replacing r_2 by $\texttt{c}\, s_2$ (resp. by $\texttt{d}\, s_2$) does: $\{\texttt{a}\, \widehat{x}\, (\texttt{c}\, \widehat{y})/_{\{x,y\}}\, \texttt{a}\, r_1\, (\texttt{c}\, s_2)\} = \{x \to r_1, y \to s_2\}$ (resp. $\{\texttt{a}\, \widehat{x}\, (\texttt{c}\, \widehat{y})/_{\{x,y\}}\, \texttt{a}\, r_1\, (\texttt{d}\, s_2)\} = \texttt{fail}$). Hence, contraction of r_2 can contribute towards obtaining a decided match, while contraction of r_1 does not.

The selection of redexes contributing towards a decided match is performed by a simultaneous structural analysis of both pattern and argument. Since $\texttt{PPC}$ allows patterns to be reduced, the selected redexes can lie inside a pattern or an argument of a preredex; and not in the body of the abstraction. Take *e.g.* $(\lambda_{\{x,y\}}\, \texttt{a}\, (\texttt{b}\, \widehat{x})\, r_1.r_2)\, (\texttt{a}\, r_3\, (\texttt{d}\, r_4))$ where every r_i is a redex. The strategy selects r_1 and r_3. Moreover, notice that contraction of r_4 is delayed since r_1 is not in matchable form (if the contractum of r_1 were *e.g.* either $\texttt{d}\, \widehat{y}$ or $\texttt{a}$, then the match w.r.t. $\texttt{d}\, r_4$ would be decided without the need of reducing r_4).

The reduction strategy $\mathcal{S}$ is defined as a function from terms to *sets of positions* of redex occurrences which should be reduced as a multistep. It is defined simultaneously with an auxiliary function $\mathcal{SM}$ from sets of symbols and pairs of terms to *pairs of sets of positions* which models the selection of contributing redexes towards a decided match.

$$
\begin{array}{rll}
\mathcal{S}(x) & := \emptyset & \\
\mathcal{S}(\widehat{x}) & := \emptyset & \\
\mathcal{S}(\lambda_\theta\ p.t) & := 1\mathcal{S}(p) & \text{if } p \notin \boldsymbol{NF} \\
\mathcal{S}(\lambda_\theta\ p.t) & := 2\mathcal{S}(t) & \text{if } p \in \boldsymbol{NF} \\
\mathcal{S}((\lambda_\theta\ p.t)u) & := \{\epsilon\} & \text{if } \{p/_\theta\ u\} \text{ decided} \\
\mathcal{S}((\lambda_\theta\ p.t)u) & := 11G \cup 2D & \text{if } \{p/_\theta\ u\} = \mathtt{wait}, \mathcal{SM}_\theta(p,u) = \langle G, D\rangle \neq \langle\emptyset,\emptyset\rangle, \\
\mathcal{S}((\lambda_\theta\ p.t)u) & := 11\mathcal{S}(p) & \text{if } \{p/_\theta\ u\} = \mathtt{wait}, \mathcal{SM}_\theta(p,u) = \langle\emptyset,\emptyset\rangle, p \notin \boldsymbol{NF} \\
\mathcal{S}((\lambda_\theta\ p.t)u) & := 12\mathcal{S}(t) & \text{if } \{p/_\theta\ u\} = \mathtt{wait}, \mathcal{SM}_\theta(p,u) = \langle\emptyset,\emptyset\rangle, p \in \boldsymbol{NF}, t \notin \boldsymbol{NF} \\
\mathcal{S}((\lambda_\theta\ p.t)u) & := 2\mathcal{S}(u) & \text{if } \{p/_\theta\ u\} = \mathtt{wait}, \mathcal{SM}_\theta(p,u) = \langle\emptyset,\emptyset\rangle, p \in \boldsymbol{NF}, t \in \boldsymbol{NF} \\
\mathcal{S}(tu) & := 1\mathcal{S}(t) & \text{if } t \text{ is not an abstraction and } t \notin \boldsymbol{NF} \\
\mathcal{S}(tu) & := 2\mathcal{S}(u) & \text{if } t \text{ is not an abstraction and } t \in \boldsymbol{NF}
\end{array}
$$

$$
\begin{array}{rll}
\mathcal{SM}_\theta(\widehat{x}, t) & := \langle\emptyset,\emptyset\rangle & \text{if } x \in \theta \\
\mathcal{SM}_\theta(\widehat{x}, \widehat{x}) & := \langle\emptyset,\emptyset\rangle & \text{if } x \notin \theta \\
\mathcal{SM}_\theta(p_1p_2, t_1t_2) & := \langle 1G_1 \cup 2G_2, 1D_1 \cup 2D_2\rangle & \text{if } t_1t_2, p_1p_2 \in \boldsymbol{MF}, \mathcal{SM}_\theta(p_i, t_i) = \langle G_i, D_i\rangle \\
\mathcal{SM}_\theta(p, t) & := \langle \mathcal{S}(p), \emptyset\rangle & \text{if } p \notin \boldsymbol{MF} \\
\mathcal{SM}_\theta(p, t) & := \langle \emptyset, \mathcal{S}(t)\rangle & \text{if } p \in \boldsymbol{MF} \ \&\ t \notin \boldsymbol{MF} \ \&\ \neg(p = \widehat{x} \ \&\ x \in \theta)
\end{array}
$$

The auxiliary function $\mathcal{SM}$ formalises the simultaneous structural analysis of the argument and pattern of a preredex. Its outcome is a pair of sets of positions, corresponding to redexes inside the pattern and argument respectively, which could contribute to turning a non decided match into a decided one. Notice the similarities between the first three clauses in the definition of $\mathcal{SM}$ and those of the definition of the matching operation (*cf.* Sec. 2).

If the LO preredex of a term is in fact a redex, then the strategy selects exactly that redex (fifth clause); if it is not a redex, and the function $\mathcal{SM}$ returns some redexes which could contribute towards a decided match, then the strategy selects them (sixth clause).

Otherwise, the preredex will never turn into a redex. Indeed, it can be proved that, given p and u such that $\{p/_\theta\ u\} = \mathtt{wait}$, if there exist p' and u' such that $p \twoheadrightarrow p'$, $u \twoheadrightarrow u'$ and $\{p'/_\theta\ u'\}$ is decided, then $\mathcal{SM}_\theta(p,u) \neq \langle\emptyset,\emptyset\rangle$. In this case the strategy looks for the LO preredex inside the components of the term (seventh, eighth and ninth clauses).

The remaining clauses in the definition of $\mathcal{S}$ formalise the focus on the LO preredex for other forms of the term.

Returning to the example t_2 given in Sec. 1, namely $(\lambda_{\{y\}}\ \mathtt{a\ b\ c}\ \widehat{y}.y)\ (\mathtt{a}\ (\mathtt{I\ c})\ (\mathtt{I\ b})\ (\mathtt{I\ a}))$, notice that the set $\{\mathtt{I\ c}, \mathtt{I\ b}\}$ is selected by the strategy $\mathcal{S}$, even if the contraction of just one redex of the set suffices to make the head match decided as explained before.

The reduction strategy $\mathcal{S}$ is complete. Formally:

▶ **Lemma 2.** *Let $t \notin \boldsymbol{NF}$. Then $\mathcal{S}(t) \neq \emptyset$, and $\mathcal{S}(t)$ only contains outermost redexes in t.*

Moreover, notice that $\mathcal{S}$ is not *outermost fair* [23]. Indeed, given $(\lambda\,\mathtt{c}\,x.s)\,\Omega$, where Ω is a non-terminating term, $\mathcal{S}$ continuously contracts Ω, even when s contains a redex.

5 The reduction strategy $\mathcal{S}$ is normalising

This section proves that $\mathcal{S}$ is normalising for PPC. Our proof is mainly inspired from [21] which proves that selecting *necessary sets* (as introduced in Sec. 1) is a sufficient condition for a reduction strategy to be normalising in a first-order setting.

$$\begin{tikzcd}
t_0 = t \arrow[r, ->>, "\Delta_0"] \arrow[d, "\mathcal{S}(t_0)"'] & u \arrow[d, equal] \\
t_1 \arrow[r, ->>, "\Delta_1"] \arrow[d, dotted, no head] & u \arrow[d, dotted, no head] \\
t_n \arrow[r, ->>, "\Delta_n"] \arrow[d, "\mathcal{S}(t_n)"'] & u \arrow[d, equal] \\
u \arrow[r, equal, "\Delta_{n+1}"] & u
\end{tikzcd}$$

Figure 4 Proof idea.

The proof proceeds as follows: given any multireduction to normal form starting from a term t_0, say $t_0 \overset{\Delta_0}{\twoheadrightarrow} u \in \boldsymbol{NF}$, we construct another multireduction starting from t_0 to normal form having the form $t_0 \xrightarrow{\mathcal{S}(t_0)} t_1 \overset{\Delta_1}{\twoheadrightarrow} u$; where $\mathcal{S}(t_0)$ is the set of redexes of t_0 chosen by the strategy $\mathcal{S}$ and the multireduction Δ_1 is strictly smaller than the original one w.r.t. a convenient well-founded ordering. Well-foundedness of the ordering entails that repeated evaluation of the set of redexes selected by the strategy $\mathcal{S}$ yields the normal form u. This is depicted in Fig. 4 where Δ_{k+1} is strictly smaller than Δ_k for all k and Δ_{n+1} is a trivial multireduction. Thus, the original multireduction Δ_0 is first transformed into $\mathcal{S}(t_0); \Delta_1$, then successively into $\mathcal{S}(t_0); \ldots; \mathcal{S}(t_k); \Delta_{k+1}$; and finally into $\mathcal{S}(t_0); \ldots; \mathcal{S}(t_n)$.

Some difficulties arise when developing such a proof for higher-order calculi like `PPC`, particularly in two aspects: it is not trivial to show that the succesive multireductions which are built during the proof have the same target; nor is it straightforward to show that the sequence of multireductions $\Delta_0, \ldots, \Delta_k, \ldots$ is strictly decreasing.

To handle the first of these problems, we use a technique consisting of postponement of certain (non relevant) redexes with respect to other (relevant) ones (our notion of relevant is however different from that appearing in [22]). As for the second problem we define a measure inspired from [21, 22]. In order to prove that the sequence $\Delta_0, \ldots, \Delta_k, \ldots$ is strictly decreasing w.r.t. this measure, we need to resort to an additional property verified by the sets of redexes selected by $\mathcal{S}$, namely that they are *non-gripping*. In the following, we formalise the concept of necessary sets of redexes, we motivate and introduce the additional non-gripping property, and finally we give a brief outline of the proof.

5.1 Necessary sets of redexes

As explained in Sec. 1, there are some terms in `PPC` which do not have any *needed* redex; this was illustrated by the term $t_2 := (\lambda_{\{y\}}$ `a b c` $\widehat{y}.y)$ (`a` (`I c`) (`I b`) (`I a`)) whose (outermost) redexes are `I c`, `I b` and `I a`: it is possible to construct a reduction sequence from t_2 to normal form which ignores either of these redexes.

To formalise the meaning of needed redex, we first resort to the following notion of ignoring (or not) a redex along a (multi)reduction: given a term t and $\mathcal{B} \subseteq \mathcal{RO}(t)$, a multistep/multireduction from t *uses* $\mathcal{B}$ iff it contracts at least one redex or one residual of a redex in $\mathcal{B}$. Formally, let t be a term, $b \in \mathcal{RO}(t)$, $\mathcal{B} \subseteq \mathcal{RO}(t)$, a multistep $t \xrightarrow{\mathcal{A}} u$ and a multireduction $t \overset{\Delta}{\twoheadrightarrow} u$. Then, $\mathcal{A}$ **uses** b iff $b \in \mathcal{A}$; Δ **uses** b iff $\Delta[k] \cap (b/\Delta[1..k-1]) \neq \emptyset$ for at least one k; $\mathcal{A}$ (resp. Δ) **uses** $\mathcal{B}$ iff it uses at least one $b \in \mathcal{B}$. Now, given a term t, a redex $a \in \mathcal{RO}(t)$ is **needed** for t iff every reduction from t to normal form uses a.

Notice that the above term t_2 has no needed redex. Now consider the *set* of redexes $\{$`I c`$,$`I b`$\}$ in the same term t_2. All reductions from t_2 to normal form must use at least one of these redexes *i.e. the set of redexes* $\{$`I c`$,$`I b`$\}$ as a whole cannot be ignored to obtain the normal form of t_2. We formalise this notion as follows: given a term t, a set $\mathcal{A} \subseteq \mathcal{RO}(t)$ is a **necessary set** for t, iff every reduction from t to normal form uses $\mathcal{A}$.

The notion of necessary set generalises that of needed redex (notice that any singleton whose only element is a needed redex is a necessary set). There is, however, an important difference: while not all terms admit a needed redex, any term admits at least one necessary set, *i.e.* the set of *all* its redexes.

The reduction strategy $\mathcal{S}$ defined in Sec. 4 selects *necessary sets* of redexes; this property turns out to be crucial to prove that $\mathcal{S}$ is normalising. Formally:

▶ **Proposition 3.** *Let t be a term such that $t \notin$ **NF**. Then $\mathcal{S}(t)$ is a necessary set for t.*

5.2 Gripping

To motivate the notion of *gripping* [16], let us consider the following example, suggested by V. van Oostrom:

$$t_5 := \left((\lambda_{\{x\}}\widehat{x}.{D\ x}^{b})\ {(\mathtt{I}\ y)}^{a_2}\right)^{a_1} \overset{\mathcal{B}}{\rightarrow} \left((\lambda_{\{x\}}\widehat{x}.x\ x)\ {(\mathtt{I}\ y)}^{a_2}\right)^{a_1} = u_5$$

where $\mathcal{A} := \{a_1, a_2\}$, $\mathcal{B} := \{b\}$ and $D := \lambda_{\{x\}}\ \widehat{x}.x\ x$. It is easy to verify that $\mathcal{A}/\mathcal{B} = \mathcal{A}$. Nevertheless, the *depth* (*cf.* Sec. 3) of $\mathcal{A}$ does change: the set $\mathcal{A}$ admits a development from u_5 requiring two contractions of $\mathtt{I}\ y$, while this is not the case for t_5; yielding $\nu(\mathcal{A}, t_5) = 2 < 3 = \nu(\mathcal{A}, u_5)$.

The notion of gripping turns out to be appropiate to explain this example. Informally, a redex b *grips* another redex a iff $a < b$ and there are occurrences of variables inside b which are bound by the abstraction of the redex a. In such a case, b-reduction may duplicate or erase those variable occurrences, thus affecting the depth of multisteps including a. In the term t_5 above, the redex b grips the redex a_1 because x, ocurring inside b, is bound by the abstraction of a_1; duplication of x explains the depth increase of $\mathcal{A}$ from u_5.

The following definition formalises the notion of gripping for $\mathtt{PPC}$: given $a, b \in \mathcal{RO}(t)$, say $t|_a = (\lambda_\theta p.s)u$, we say that b **grips** a iff $\{p/_\theta\ u\} \neq \mathtt{fail}$, $a12 \leq b$ and $\mathtt{fv}(t|_b) \cap \theta \neq \emptyset$. Given $\mathcal{A}, \mathcal{B} \subseteq \mathcal{RO}(t)$, we say that $\mathcal{B}$ **grips** $\mathcal{A}$ iff there exist $b \in \mathcal{B}$ and $a \in \mathcal{A}$ such that b grips a.

In addition, we define a set $\mathcal{B} \subseteq \mathcal{RO}(t)$ to be **non-gripping** in t (or just **non-gripping** if t is clear from the context) iff for any multireduction Ψ such that $t \overset{\Psi}{\twoheadrightarrow} u$, $\mathcal{B}/\Psi$ does not grip $\mathcal{RO}(u)$. Notice that when $\mathcal{B}$ is non-gripping all its residuals are.

The reduction strategy $\mathcal{S}$ defined in Sec. 4 selects *non-gripping* sets of redexes, formally

▶ **Proposition 4.** *Let t be a term. Then $\mathcal{S}(t)$ is non-gripping in t.*

Proof. (sketch) One first proves that $t \overset{\Psi}{\twoheadrightarrow} u$, $a \in \mathcal{RO}(u)$, $b \in \mathcal{S}(t)/\Psi$ and $a < b$ imply a is a matching failure. This can be done by induction on the size of t, by considering the different cases in the (mutually recursive) definitions of $\mathcal{S}$ and $\mathcal{SM}$.

Now, let $t \overset{\Psi}{\twoheadrightarrow} u$. To prove that $\mathcal{S}(t)$ is non-gripping in t we need to show that $\mathcal{S}(t)/\Psi$ does not grip $\mathcal{RO}(u)$. Suppose $b \in \mathcal{S}(t)/\Psi$, $a \in \mathcal{RO}(u)$ and b grips a. Then in particular $a < b$ and a is not a matching failure. But the previous observation entails that a is a matching failure which leads to a contradiction. ◀

This property allows to resort to gripping in order to guarantee that the depth of sets of redexes is stable under reduction in the cases in which this stability is needed in the normalisation proof. Another approach, distinguishing between *essential* and *inessential* sets of redexes, is taken in [22].

5.3 The measure

A measure on multireductions is defined by using the notion of *depth* for multisteps, denoted by ν and defined at the end of Sec. 3. Given $\Delta = \Delta[1..n]$ and $t_{i-1} \overset{\Delta[i]}{\twoheadrightarrow} t_i$ for all i, we define $\chi(\Delta, t_0)$ as the n-tuple $\langle \nu(\Delta[n], t_{n-1}), \dots, \nu(\Delta[1], t_0)\rangle$; the lexicographic order is used to compare (measures of) multireductions. Notice that this (well-founded) ordering allows only to compare multireductions having the same length; the minimal elements being all the n-uples of the form $\langle 0, \dots, 0\rangle$. Another observation is that whenever $\chi(\Delta, t) < \chi(\Gamma, s)$ then for all multireductions $t' \overset{\Pi}{\twoheadrightarrow} t$, $s' \overset{\Psi}{\twoheadrightarrow} s$ having the same length $\chi(\Pi; \Delta, t') < \chi(\Psi; \Gamma, s')$ holds.

As remarked in [22], the measure used in [21], based on sizes of multisteps rather than depths, is not well-suited for a higher-order setting.

Returning to the key of the normalisation proof (*cf.* beginning of Sec. 5), the definition of Δ_{k+1} based on Δ_k and $\mathcal{S}(t_k)$ must verify $\chi(\Delta_{k+1}, t_{k+1}) < \chi(\Delta_k, t_k)$.

In order to further describe how Δ_{k+1} will be defined, and particularly how the notion of gripping will be applied to guarantee that the measure actually decreases, some additional notions are needed. Given a term t and $\mathcal{B} \subseteq \mathcal{RO}(t)$, a multistep/multireduction from t is *free* from $\mathcal{B}$ iff it does not contract any redex equal to or below a redex (or residual of a redex) in $\mathcal{B}$ (so it only contracts redexes lying above or disjoint with $\mathcal{B}$ and its residuals); a multistep from t is *dominated* by $\mathcal{B}$ iff all their redexes lie below some redex in $\mathcal{B}$. Formally, let t be a term, $a, b \in \mathcal{RO}(t)$, $\mathcal{A}, \mathcal{B} \subseteq \mathcal{RO}(t)$, and a multireduction $t \overset{\Delta}{\twoheadrightarrow} u$. Then,

- $\mathcal{A}$ is **free** from $\mathcal{B}$ iff $\mathcal{A} \cap \mathcal{B} = \emptyset$ and there does not exist $a \in \mathcal{A}$ and $b \in \mathcal{B}$ such that $b < a$.
- Δ is **free** from $\mathcal{B}$ iff $\Delta[k]$ is free from $\mathcal{B}/\Delta[1..k-1]$ for all k.
- a is **dominated** by $\mathcal{B}$ iff $a \notin \mathcal{B}$ and $\exists b \in \mathcal{B}$ s.t. $b < a$.
- $\mathcal{A}$ is **dominated** by $\mathcal{B}$ iff $\forall a \in \mathcal{A}$, a is dominated by $\mathcal{B}$.

For example, consider the following term

$$r_1\ \ r_2\ \ (\mathtt{I}\ (\,(\lambda_{\{x\}}\ \mathtt{a}\ \widehat{x}.r_3)\ (\mathtt{a}\ \ r_4)\,)^{r_6}\,)^{r_5}\ )$$

where every r_i is a redex, and $\mathcal{B} = \{r_1, r_6\}$. Then the set $\{r_2, r_5\}$ is free from $\mathcal{B}$, $\{r_3, r_4\}$ is dominated by $\mathcal{B}$, and $\{r_2, r_3\}$ is neither free from nor dominated by $\mathcal{B}$.

Notice that $\mathcal{A}$ free from $\mathcal{B}$ and $\mathcal{C}$ dominated by $\mathcal{B}$ imply $\mathcal{A}$ free from $\mathcal{C}$.

Returning to the example in Sec. 5.2 notice that $\mathcal{A}$ is free from $\mathcal{B}$ and $\mathcal{A}/\mathcal{B} = \mathcal{A}$, however, as remarked before, $\nu(\mathcal{A}, t_5) < \nu(\mathcal{A}, u_5)$. A free set of redexes is always preserved by reduction; moreover, gripping explains all the cases in which the depth changes. Formally,

▶ **Lemma 5.** *Let $\mathcal{A}, \mathcal{B} \subseteq \mathcal{RO}(t)$ s.t. $\mathcal{A}$ is free from $\mathcal{B}$. Then $\mathcal{A}/\mathcal{B} = \mathcal{A}$.*

▶ **Lemma 6.** *Let $\mathcal{A}, \mathcal{B} \subseteq \mathcal{RO}(t)$ s.t. $\mathcal{A}$ is free from $\mathcal{B}$ and $t \overset{\mathcal{B}}{\rightarrow} s$. If $\mathcal{B}$ does not grip $\mathcal{A}$, then $\nu(\mathcal{A}, t) = \nu(\mathcal{A}, s)$.*

5.4 The normalisation proof

In this seciton we give a proof of the main result of the paper, namely that the strategy $\mathcal{S}$ is normalising. The proof is based on the ideas described at the beginning of Sec. 5. The main auxiliary results used in the proof are also included. They formalise the construction of Δ_{k+1} (*cf.* Fig. 4), in their statements $\mathcal{B}$ can be considered to be (some residual of) $\mathcal{S}(t_k)$ and $\mathcal{A}, \mathcal{C}, \Delta$ to be Δ_k or parts of it.

▶ **Lemma 7.** *Let $\mathcal{B}, \mathcal{C} \subseteq \mathcal{RO}(t)$ s.t. $t \overset{\mathcal{C}}{\circ\!\!\rightarrow} u$, and $\mathcal{A} \subseteq \mathcal{RO}(u)$ s.t. $\mathcal{C}$ is dominated by $\mathcal{B}$, $\mathcal{A}$ is free from $\mathcal{B}/\mathcal{C}$, and $\mathcal{B}$ is non-gripping. Then $\mathcal{A} \subseteq \mathcal{RO}(t)$, $\mathcal{A}$ is free from $\mathcal{B}$, and $\nu(\mathcal{A}, t) = \nu(\mathcal{A}, u)$.*

Proof. (sketch) To obtain $\mathcal{A} \subseteq \mathcal{RO}(t)$ and $\mathcal{A}$ free from $\mathcal{B}$, we reason by induction on $\nu(\mathcal{C}, t)$; so let $c \in \mathcal{C}$ and $\delta = c; \delta'$ be a complete development of $\mathcal{C}$ (so δ' is a complete development of $\mathcal{C}/c$), *i.e.* $t \xrightarrow{c} s \overset{\mathcal{C}/c}{\circ\!\!\rightarrow} u$. It is not difficult to prove that $\mathcal{C}$ dominated by $\mathcal{B}$ and $c \in \mathcal{C}$ imply $\mathcal{C}/c$ dominated by $\mathcal{B}/c$, allowing to use the *i.h.* on $\mathcal{C}/c$ to obtain $\mathcal{A} \subseteq \mathcal{RO}(s)$ and $\mathcal{A}$ free from $\mathcal{B}/c$. Now, for any $a \in \mathcal{A} \subseteq \mathcal{RO}(s)$, it can be proved (by contradiction) that $a \in \mathcal{RO}(t)$ and a is free from $\mathcal{B}$. Thus, $\mathcal{A}$ is free from $\mathcal{B}$. Noticing that $\mathcal{B}$ being non-gripping (so $\mathcal{B}$ does not grip $\mathcal{A}$), $\mathcal{A}$ free from $\mathcal{B}$ and $\mathcal{C}$ dominated by $\mathcal{B}$ imply $\mathcal{C}$ does not grip $\mathcal{A}$, depth stability follows from Lem. 6. ◀

▶ **Lemma 8.** *Let $t \overset{\mathcal{C}}{\circ\!\!\rightarrow} s \overset{\Delta}{\circ\!\!\twoheadrightarrow} u$ and $\mathcal{B} \subseteq \mathcal{RO}(t)$ s.t. $\mathcal{B}$ is non-gripping, $\mathcal{C}$ is dominated by $\mathcal{B}$, Δ is free from $\mathcal{B}/\mathcal{C}$, and $\mathcal{B}/(\mathcal{C}; \Delta) = \emptyset$. Then $t \overset{\Delta}{\circ\!\!\twoheadrightarrow} u$, Δ is free from $\mathcal{B}$, $\mathcal{B}/\Delta = \emptyset$ and $\chi(\Delta, t) = \chi(\Delta, s)$.*

Proof. We proceed by induction on the size of Δ.

Assume $\Delta = \texttt{nil}$. Then $\mathcal{B}/(\mathcal{C}; \Delta) = \mathcal{B}/\mathcal{C} = \emptyset$. We first show $\mathcal{B} = \emptyset$. Indeed, suppose that $\mathcal{B} \neq \emptyset$, let b be a minimal element of $\mathcal{B}$ w.r.t. the prefix order. It is straightforward to verify that $\{b\}$ is free from $\mathcal{C}$ (since $\mathcal{C}$ is dominated by $\mathcal{B}$), then Lem. 5 yields $\{b\}/\mathcal{C} = \{b\}$, contradicting $\mathcal{B}/\mathcal{C} = \emptyset$. Then $\mathcal{B} = \emptyset$, therefore $\mathcal{C} = \emptyset$, again since $\mathcal{C}$ is dominated by $\mathcal{B}$, hence $t = s = u$ and the conclusions are straightforward.

If $\Delta \neq \texttt{nil}$, then consider $t \overset{\mathcal{C}}{\circ\!\!\rightarrow} s \overset{\Delta[1]}{\circ\!\!\rightarrow} w \overset{\Delta[2..n]}{\circ\!\!\twoheadrightarrow} u$. Lem. 7 gives $\Delta[1] \subseteq \mathcal{RO}(t)$, $\Delta[1]$ is free from $\mathcal{B}$ and $\nu(\Delta[1], t) = \nu(\Delta[1], s)$; moreover, since $\Delta[1]$ is free from $\mathcal{B}$ and $\mathcal{C}$ is dominated by $\mathcal{B}$, then $\Delta[1]/\mathcal{C} = \Delta[1]$ (*cf.* Lem. 5), so $(\mathcal{C}; \Delta[1])$ and $(\Delta[1]; (\mathcal{C}/\Delta[1]))$ are two complete developments of $\Delta[1] \cup \mathcal{C}$. Hence, Prop. 1 implies that $t \overset{\Delta[1]}{\circ\!\!\rightarrow} s' \overset{\mathcal{C}/\Delta[1]}{\circ\!\!\rightarrow} w \overset{\Delta[2..n]}{\circ\!\!\twoheadrightarrow} u$.

In order to apply the *i.h.* we need to verify the corresponding hypotheses. By a patient analysis on residuals and positions we obtain that $\mathcal{C}/\Delta[1]$ is dominated by $\mathcal{B}/\Delta[1]$. Moreover, $\mathcal{B}$ non-gripping implies $\mathcal{B}/\Delta[1]$ non-gripping, and the remaining conditions can be easily obtained by noticing that $\mathcal{B}/(\Delta[1]; (\mathcal{C}/\Delta[1])) = \mathcal{B}/(\mathcal{C}; \Delta[1])$.

Therefore the *i.h.* can be applied, obtaining $s' \overset{\Delta[2..n]}{\circ\!\!\twoheadrightarrow} u$, $\Delta[2..n]$ is free from $\mathcal{B}/\Delta[1]$, $\mathcal{B}/(\Delta[1]; \Delta[2..n]) = \emptyset$, and $\chi(\Delta[2..n], s') = \chi(\Delta[2..n], w)$. We conclude by combining these results with those obtained in the first paragraph. ◀

▶ **Lemma 9.** *Let $t \overset{\Delta}{\circ\!\!\twoheadrightarrow} u$ and $\mathcal{B} \in \mathcal{RO}(t)$ s.t. $\Delta = \Delta[1..n]$, $\mathcal{B}$ is non-gripping, Δ does not use $\mathcal{B}$ and $\mathcal{B}/\Delta = \emptyset$. Then there exists a multireduction $\Gamma = \Gamma[1..n]$ such that $t \overset{\Gamma}{\circ\!\!\twoheadrightarrow} u$, Γ is free from $\mathcal{B}$, $\mathcal{B}/\Gamma = \emptyset$ and $\chi(\Gamma, t) \leq \chi(\Delta, t)$.*

Proof. We proceed by induction on n.

If $n = 0$ then $\Delta = \emptyset$, therefore it suffices to take $\Gamma = \emptyset$.

If $n > 0$ then we consider $t \overset{\Delta[1]}{\circ\!\!\rightarrow} s \overset{\Delta[2..n]}{\circ\!\!\twoheadrightarrow} u$. By observing that $\mathcal{B}/\Delta[1]$ is non-gripping we can use the *i.h.* on $t \overset{\Delta[2..n]}{\circ\!\!\twoheadrightarrow} u$, thus obtaining a multireduction $\Gamma_1 = \Gamma_1[1..n-1]$ such that $s \overset{\Gamma_1}{\circ\!\!\twoheadrightarrow} u$, Γ_1 is free from $\mathcal{B}/\Delta[1]$, $(\mathcal{B}/\Delta[1])/\Gamma_1 = \emptyset$ and $\chi(\Gamma_1, s) \leq \chi(\Delta[2..n], s)$.

We now define $\Delta[1]^F := \{a \in \Delta[1] \text{ s.t. } \nexists b \in \mathcal{B} \,.\, b < a\}$ and $\Delta[1]^D := (\Delta[1] \setminus \Delta[1]^F)/\Delta[1]^F$, then $t \overset{\Delta[1]^F}{\circ\!\!\rightarrow} t' \overset{\Delta[1]^D}{\circ\!\!\rightarrow} s \overset{\Gamma_1}{\circ\!\!\twoheadrightarrow} u$ for some term t'. It is easy to check that $\Delta[1]^F$ is free from $\mathcal{B}$ and $\Delta[1] \setminus \Delta[1]^F$ is dominated by $\mathcal{B}$ just by definition ($\Delta[1]$ does not use $\mathcal{B}$), and it can be proved that $\Delta[1]^D$ is dominated by $\mathcal{B}/\Delta[1]^F$. Moreover $\mathcal{B}/\Delta[1]^F$ is non-gripping. Finally, $\mathcal{B}/\Delta[1] = (\mathcal{B}/\Delta[1]^F)/\Delta[1]^D$ by Prop. 1; consequently Γ_1 is free from $(\mathcal{B}/\Delta[1]^F)/\Delta[1]^D$ and $(\mathcal{B}/\Delta[1]^F)/(\Delta[1]^D;\Gamma_1) = \emptyset$.

Therefore we can use Lem. 8 on $t' \overset{\Delta[1]^D}{\longrightarrow} s \overset{\Gamma_1}{\twoheadrightarrow} u$, thus obtaining that $t' \overset{\Gamma_1}{\twoheadrightarrow} u$, Γ_1 is free from $\mathcal{B}/\Delta[1]^F$, $(\mathcal{B}/\Delta[1]^F)/\Gamma_1 = \emptyset$ and $\chi(\Gamma_1, t') = \chi(\Gamma_1, s) \leq \chi(\Delta[2..n], s)$.

We can conclude by taking $\Gamma := \Delta[1]^F;\Gamma_1$; notice that $\nu(\Delta[1]^F, t) \leq \nu(\Delta[1], t)$. ◀

▶ **Proposition 10.** *Let $t \overset{\Delta}{\twoheadrightarrow} u$ and $\mathcal{B} \subseteq \mathcal{RO}(t)$ s.t. $\mathcal{B}$ is non-gripping, Δ does not use $\mathcal{B}$, $\mathcal{B}/\Delta = \emptyset$ and $t \overset{\mathcal{B}}{\longrightarrow} s$. Then $\exists$ Γ s.t. $s \overset{\Gamma}{\twoheadrightarrow} u$ and $\chi(\Gamma, s) \leq \chi(\Delta, t)$.*

Proof. Let us say $\Delta = \Delta[1..n]$. Lem. 9 yields the existence of some $\Gamma = \Gamma[1..n]$ such that $t \overset{\Gamma}{\twoheadrightarrow} u$, Γ is free from $\mathcal{B}$, $\mathcal{B}/\Gamma = \emptyset$ and $\chi(\Gamma, t) \leq \chi(\Delta, t)$. Let us define $t_0 := t$, $t_n := u$ and $t_{i-1} \overset{\Gamma[i]}{\longrightarrow} t_i$ for all $i \leq n$. Notice that Γ being free from $\mathcal{B}$ implies that $\Gamma[i]/(\mathcal{B}/\Gamma[1..i-1]) = \Gamma[i]$ for all i, *cf.* Lem. 5. Therefore, we can build the following diagram

$$\begin{tikzcd}
t \arrow[r, "{\Gamma[1]}"] \arrow[d, "\mathcal{B}"'] & t_1 \arrow[r, "{\Gamma[2]}"] \arrow[d, "{\mathcal{B}/\Gamma[1]}"'] & t_2 \arrow[d, "{\mathcal{B}/\Gamma[1..2]}"'] & \cdots & t_{n-1} \arrow[r, "{\Gamma[n]}"] \arrow[d, "{\mathcal{B}/\Gamma[n-1]}"'] & u \arrow[d, equal, "\mathcal{B}/\Gamma=\emptyset"'] \\
s \arrow[r, "{\Gamma[1]}"'] & s_1 \arrow[r, "{\Gamma[2]}"'] & s_2 & \cdots & s_{n-1} \arrow[r, "{\Gamma[n]}"'] & u
\end{tikzcd}$$

where for all i,

Lem. 6 yields $\nu(\Gamma[i], t_{i-1}) = \nu(\Gamma[i], s_{i-1})$ since $\mathcal{B}$ non-gripping implies that $\mathcal{B}/\Gamma[1..i-1]$ does not grip $\Gamma[i]$. We conclude by observing that $\chi(\Gamma, s) = \chi(\Gamma, t) \leq \chi(\Delta, t)$. ◀

▶ **Proposition 11.** *Let $t \overset{\Delta}{\twoheadrightarrow} u$ and $\mathcal{B} \subseteq \mathcal{RO}(t)$, s.t. $\mathcal{B}$ is non-gripping, Δ uses $\mathcal{B}$, $\mathcal{B}/\Delta = \emptyset$ and $t \overset{\mathcal{B}}{\longrightarrow} s$. Then $\exists$ Γ s.t. $s \overset{\Gamma}{\twoheadrightarrow} u$ and $\chi(\Gamma, s) < \chi(\Delta, t)$.*

Proof. Let us say $\Delta = \Delta[1..n]$, $t_0 := t$, $t_n := u$ and $t_{i-1} \overset{\Delta[i]}{\longrightarrow} t_i$ for all $i \leq n$. Since Δ uses $\mathcal{B}$, there exists some $\Delta[m]$ being the last step of Δ using (the corresponding residual of) $\mathcal{B}$. Formally, if $\mathcal{B}' := \mathcal{B}/\Delta[1..m-1]$, then $\Delta[m]^1 := \Delta[m] \cap \mathcal{B}' \neq \emptyset$ and $\Delta[m+1..n]$ does not use $\mathcal{B}/\Delta[1..m]$. Additionally, let $\Delta[m]^2 := (\Delta[m] \setminus \Delta[m]^1)/\Delta[m]^1$.

We can build the following diagram

$$\begin{tikzcd}
t \arrow[r, two heads, "{\Delta[1..m-1]}"] \arrow[d, "\mathcal{B}"'] & t_{m-1} \arrow[r, "{\Delta[m]^1}"] \arrow[d, "\mathcal{B}'"'] & t'_m \arrow[r, "{\Delta[m]^2}"] \arrow[d, "{\mathcal{B}'/\Delta[m]^1}"'] & t_m \arrow[r, two heads, "{\Delta[m+1..n]}"] & u \\
s & s_{m-1} \arrow[r, equal] & s_{m-1} & &
\end{tikzcd}$$

since $\Delta[m]^1/\mathcal{B}' = \emptyset$.

Assume the existence of some $b \in \Delta[m]^2 \cap (\mathcal{B}'/\Delta[m]^1)$, this would imply that $b \in b'/\Delta[m]^1$ such that $b' \in (\Delta[m] \setminus \Delta[m]^1) \cap \mathcal{B}'$ since the ancestor of a redex is unique in `PPC`, contradicting the definition of $\Delta[m]^1$.

Therefore $\Delta' := \Delta[m]^2; \Delta[m+1..n]$ does not use $\mathcal{B}'/\Delta[m]^1$, hence Prop. 10 can be used to obtain some $\Pi = \Pi[1..n-m+1]$ such that $s_{m-1} \overset{\Pi}{\twoheadrightarrow} u$ and $\chi(\Pi, s_{m-1}) \leq \chi(\Delta', t'_m) < \chi(\Delta[m..n], t_{m-1})$.

We now define Γ as follows: $\Gamma[i] := \Delta[i]/(\mathcal{B}/\Delta[1..i-1])$ if $1 \leq i \leq m-1$, and $\Gamma[i] := \Pi[i-m+1]$ if $m \leq i \leq n$. We remark that Prop. 1 implies that $s \overset{\Gamma[1..m-1]}{\twoheadrightarrow} s_{m-1}$, thus Γ is well-defined. Moreover, the definition of the measure for multireductions by means of a *reversed* order implies that $\chi(\Pi, s_{m-1}) < \chi(\Delta[m..n], t_{m-1})$ is a sufficient condition to obtain $\chi(\Gamma, s) < \chi(\Delta, t)$. ◀

▶ **Theorem 12.** *The reduction strategy $\mathcal{S}$ is normalising.*

Proof. Let t_0 be a normalising term in `PPC`, then there exists some Δ_0 such that $t_0 \overset{\Delta_0}{\twoheadrightarrow} u$ and $u \in \boldsymbol{NF}$. We proceed by induction on $\chi(\Delta_0, t_0)$, using the well-founded ordering in Sec. 5.3.

If $t_0 \in \boldsymbol{NF}$ there is nothing to prove. Otherwise, Lem. 2 guarantees that $\mathcal{S}(t_0) \neq \emptyset$. Let $t_0 \overset{\mathcal{S}(t_0)}{\rightarrow} t_1$. Then Δ_0 uses $\mathcal{S}(t_0)$ and $\mathcal{S}(t_0)$ is non-gripping by Prop. 3 and Prop. 4 respectively; moreover, $u \in \boldsymbol{NF}$ implies $\mathcal{S}(t_0)/\Delta_0 = \emptyset$. Then Prop. 11 yields the existence of a multireduction Δ_1 s.t. $t_1 \overset{\Delta_1}{\twoheadrightarrow} u$ and $\chi(\Delta_1, t_1) < \chi(\Delta_0, t_0)$. We conclude by the *i.h.* ◀

As a final remark, notice that the construction of Δ_{k+1} from Δ_k and $\mathcal{S}(t_k)$ in the proof of Prop. 11 combines two different kinds of projections: one based on residuals (*cf.* Prop. 1), the other based in the notions of free and dominated sets of redexes.

6 Conclusions and further work

We study normalisation strategies for `PPC`, a dynamic pattern calculus equipped with *matching failure*. Its semantics induces a parallel-or-like, non-sequential behaviour which hinders the development of normalising strategies, particularly since it is not a FO system nor is it clear how it may be encoded in terms of established HO rewriting formalisms (eg. HRS [18], CRS [14]).

Building on ideas from [21] developed for FO systems, we propose a notion of *necessary set* of redexes for a HO language. Repeated contraction of necessary sets is shown to normalise a term *provided* that they are also *non-gripping* [16]. We introduce an inductively defined strategy that, given a term t, selects a necessary set of redexes for t which is also non-gripping, and moreover bounded by the set of outermost redexes. The strategy collapses to LO when the λ-calculus is encoded in `PPC`.

We think that our normalisation proof could be adapted to other (HO) calculi, and even to families of calculi defined in some general HO formalism, particularly since necessary and gripping sets are specified in a quite general way. Another research direction is to adopt a completely axiomatic approach, *e.g.* Abstract Rewriting Systems as defined in [16].

An encoding of `PPC` into some HO formalism, such as those mentioned above, could yield interesting insights on the possible transfer of the normalisation results of [23] and [22] from HORS to our framework.

Further avenues of research we intend to pursue include implementing an interpreter based on our strategy and devising even more refined strategies, in the sense of selecting smaller sets of redexes.

Acknowledgements To Vincent van Oostrom (for having pointed out a mistake in a previous version of this work), Femke van Raamsdonk and the anonymous referees (for helpful remarks). This work was partially funded by the Digiteo-IdF project MoDy and the French-Argentinian Laboratory in Computer Science **INFINIS**.

References

1 S. Antoy and A. Middeldorp. A sequential reduction strategy. *TCS*, 165(1):75–95, 1996.

2 F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge Univ. Press, 1998.

3 T. Balabonski. On the implementation of dynamic patterns. In *HOR*, EPTCS 49, pages 16–30, Edinburgh, UK, 2010.

4 T. Balabonski. Optimality for dynamic patterns: Extended abstract. In *PPDP*, pages 16–30. ACM, 2010.

5 H.P. Barendregt, R. Kennaway, J-W. Klop, and M. Ronan Sleep. Needed reduction and spine strategies for the lambda calculus. *I & C*, 75(3):191–231, 1987.

6 H. Cirstea and C. Kirchner. The rewriting calculus - Part I and Part II. *Logic Journal of the IGPL*, 9(3), 2001.

7 J. Glauert, R. Kennaway, and Z. Khasidashvili. Stable results and relative normalization. *JLC*, 10(3):323–348, 2000.

8 G. Huet and J-J Lévy. Computations in orthogonal rewriting systems - Parts I and II. In *Computational Logic, Essays in Honor of Alan Robinson*, pages 395–443. MIT Press, 1991.

9 B. Jay. *Pattern Calculus: Computing with Functions and Structures.* Springer, 2009.

10 B. Jay and D. Kesner. Pure pattern calculus. In *ESOP*, *LNCS* 3924, pages 100–114. Springer, 2006.

11 B. Jay and D. Kesner. First-class patterns. *JFP*, 19(2):191–225, 2009.

12 W. Kahl. Basic pattern matching calculi: A fresh view on matching failure. In *FLOPS*, *LNCS* 2998, pages 276–290. Springer, 2004.

13 D. Kesner, C. Lombardi, and A. Ríos. Standardisation for constructor based pattern calculi. In *HOR*, EPTCS 49, pages 58–72, Edinburgh, UK, 2010.

14 J-W. Klop. Combinatory Reduction Systems. *Mathematical Centre Tracts* 127. PhD thesis, University Amsterdam, 1980.

15 J-W. Klop, V. van Oostrom, and R. de Vrijer. Lambda calculus with patterns. *TCS*, 398(1-3):16–31, 2008.

16 P-A. Melliès. *Description abstraite des Systèmes de Réécriture.* PhD thesis, Université Paris VII, 1996.

17 P-A. Melliès. Axiomatic rewriting theory II: the $\lambda\sigma$-calculus enjoys finite normalisation cones. *JLC*, 10(3):461–487, 2000.

18 T. Nipkow. Higher-Order Critical Pairs. In *LICS*, pages 342–349, IEEE. 1991.

19 M. J. O'Donnell. *Computing in Systems Described by Equations*, *LNCS* 58. Springer, 1977.

20 S. L. Peyton-Jones. *The Implementation of Functional Programming Languages.* Prentice-Hall, Inc., 1987.

21 R. C. Sekar and I. V. Ramakrishnan. Programming in equational logic: Beyond strong sequentiality. *I & C*, 104(1):78–109, 1993.

22 V. van Oostrom. Normalisation in weakly orthogonal rewriting. In *RTA*, *LNCS* 1631, pages 60–74. Springer, 1999.

23 F. van Raamsdonk. Outermost-fair rewriting. In *TLCA*, *LNCS* 1210, pages 284–299. Springer, 1997.

A Semantic Proof that Reducibility Candidates entail Cut Elimination

Denis Cousineau[1] and Olivier Hermant[2]

1 INRIA-Saclay, France
denis@cousineau.eu
2 ISEP, France
olivier.hermant@isep.fr

Abstract

Two main lines have been adopted to prove the cut elimination theorem: the syntactic one, that studies the process of reducing cuts, and the semantic one, that consists in interpreting a sequent in some algebra and extracting from this interpretation a cut-free proof of this very sequent.

A link between those two methods was exhibited by studying in a semantic way, syntactical tools that allow to prove (strong) normalization of proof-terms, namely reducibility candidates. In the case of deduction modulo, a framework combining deduction and rewriting rules in which theories like Zermelo set theory and higher order logic can be expressed, this is obtained by constructing a reducibility candidates valued model. The existence of such a *pre*-model for a theory entails strong normalization of its proof-terms and, by the usual syntactic argument, the cut elimination property.

In this paper, we strengthen this gate between syntactic and semantic methods, by providing a full semantic proof that the existence of a pre-model entails the cut elimination property for the considered theory in deduction modulo. We first define a new simplified variant of reducibility candidates *à la* Girard, that is sufficient to prove weak normalization of proof-terms (and therefore the cut elimination property). Then we build, from some model valued on the pre-Heyting algebra of those WN reducibility candidates, a regular model valued on a Heyting algebra on which we apply the usual soundness/strong completeness argument.

Finally, we discuss further extensions of this new method towards normalization by evaluation techniques that commonly use Kripke semantics.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases cut elimination, reducibility candidates, (pre-)Heyting algebras.

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.133

Category Regular Research Paper

1 Introduction

The cut elimination theorem [15] is a central result in proof theory and type theory. From a proof theorist's point of view, it implies the consistency of the considered logical framework, as well as other nice results like the disjunction property, the witness property or the subformula property. On the other side, the type theorist is more interested in the cut elimination process itself, and in its termination. Those two different interests led to two distinct main lines of showing cut elimination, namely the semantic and the syntactic methods.

The semantic methods [17, 4, 18] use the soundness/strong completeness paradigm: first show that if we have a proof of A under the hypothesis Γ, then every model of Γ, valued on a Heyting algebra, is a model of A, and then show that if the latter holds, then we can build a cut-free proof of A when assuming Γ.

A modern variant of the syntactic method uses the Curry-Howard correspondence and Tait-Girard's reducibility method [16], in order to prove normalization of β-reduction on proof-terms (that syntactically entails cut elimination).

The logical framework we shall work in is Deduction modulo [11]. It is a generic way to integrate computation rules into a deduction system, in our case natural deduction. In this logical framework, theories are expressed via rewrite rules on first order terms and propositions, instead of axioms. One can express, only with rewrite rules, both theories that satisfy the cut elimination property (such as Zermelo set theory [12], Peano's arithmetic [14] or higher-order logic [10, 13]) and theories that do not. One particularity of this framework is that all theories (expressed only with rewrite rules) satisfying the cut elimination property are consistent: if there is no axiom a cut-free proof always ends with an introduction rule, and one cannot prove False with a cut-free proof. Hence, the cut elimination property entails that False is unprovable, which is not true in presence of axioms (consider, for instance, the theory containing the only axiom False. It enjoys cut elimination but it is not consistent).

A first link between the semantic and syntactic methods to prove cut elimination was made by defining reducibility candidates for deduction modulo [13] as a model, and to show that this model has a *pre*-Heyting algebra structure [9] (a Heyting algebra in which the order is replaced by a pre-order). It can be shown, with the usual (syntactic) reducibility arguments, that having such a (*pre*-)model entails strong normalization and therefore cut elimination for a theory in deduction modulo.

In this paper, we strengthen this gate between syntactic and semantic methods, by providing a full semantic proof that the existence of a pre-model entails the cut elimination property for the considered theory in deduction modulo. Since our goal is cut elimination, we consider weak normalization rather than strong normalization. We define a new simplified variant of reducibility candidates à la Girard for weak normalization, which is a first contribution of our work. We give those reducibility candidates a pre-Heyting algebra structure. Then we build, from a given model valued on that pre-Heyting algebra, a regular model valued on a Heyting algebra, that we can use to prove cut elimination with the usual soundness/strong completeness argument. This is the second contribution of our work.

Many results have been obtained in the direction we follow, especially in the normalization by evaluation approach [1, 2, 3, 5, 6], based on a Kripke-like semantic structure. In this paper, the proof of cut elimination generates, in some cases, also a normalization by evaluation algorithm, but with respect to the standard Heyting semantics notion.

We first introduce Deduction modulo in Sec. 2. Sec. 3 is devoted to semantics (Heyting and pre-Heyting algebras) and to the specific pre-Heyting algebra of reducibility candidates à la Girard for weak normalization.Then we prove, in Sec. 4 that we can extract from a model valued on that precise pre-Heyting algebra, a model valued on a Heyting algebra that allows to prove semantically cut elimination (Sec. 5). We finally discuss further extensions of the present work, especially concerning normalization by evaluation algorithms it can raise.

2 Deduction modulo

Natural Deduction modulo [10] is an extension of Natural Deduction with rewrite rules on terms and propositions. As in Natural Deduction, a theory is first defined by a language in first order logic [20], composed of a set of variables, a set of function symbols and a set of predicate symbols (all symbols given with their arities). Formulæ are then built-up from predicates (called *atomic formulæ*), the usual connectives $\Rightarrow$, $\wedge$, $\vee$ and the quantifiers $\forall$ and $\exists$. Given a language in predicate logic, a theory is defined not by axioms but by rewrite rules on terms and formulæ. In this paper, we shall not focus on how to define such a rewrite system, we will only consider the congruence relation on terms and formulæ it generates. The only mandatory property is that the congruence relation has to be non-confusing, *i.e.* two formulæ with different top connectives (or quantifiers) cannot be congruent (cut elimination does not entail consistency for confusing theories, since a cut-free proof does not necessarily end with an introduction rule). The principle of deduction modulo is to adapt the typing/deduction rules of natural deduction, giving the ability to replace a formula by an equivalent one, at each step of a typing derivation, as detailed in Fig. 1. We use the Curry-Howard correspondence to express proof-terms of deduction modulo. Those proof-terms can contain both term variables (written $x, y, \ldots$, given by the language in predicate logic) and proof variables (written $\alpha, \beta, \ldots$). In the same way, terms are written $t, u, \ldots$ while proof-terms are written $\pi, \rho, \ldots$. Typing contexts Γ are sequences of labeled formulæ: $\alpha_1 : A_1, \cdots, \alpha_n : A_n$.

$$\frac{}{\Gamma \vdash \alpha : B}\ \text{axiom, if } \alpha : A \in \Gamma \text{ and } A \equiv B$$

$$\frac{\Gamma, \alpha : A \vdash \pi : B}{\Gamma \vdash \lambda\alpha.\pi : C}\ \Rightarrow\text{-intro, if } C \equiv A \Rightarrow B$$

$$\frac{\Gamma \vdash \pi : C \qquad \Gamma \vdash \pi' : A}{\Gamma \vdash (\pi\ \pi') : B}\ \Rightarrow\text{-elim, if } C \equiv A \Rightarrow B$$

$$\frac{\Gamma \vdash \pi : A \qquad \Gamma \vdash \pi' : B}{\Gamma \vdash \langle \pi, \pi' \rangle : A \wedge B}\ \wedge\text{-intro, if } C \equiv A \wedge B$$

$$\wedge\text{-elim1, if } C \equiv A \wedge B\ \frac{\Gamma \vdash \pi : C}{\Gamma \vdash fst(\pi) : A} \qquad \frac{\Gamma \vdash \pi : C}{\Gamma \vdash snd(\pi) : B}\ \wedge\text{-elim2, if } C \equiv A \wedge B$$

$$\vee\text{-intro1, if } C \equiv A \vee B\ \frac{\Gamma \vdash \pi : A}{\Gamma \vdash i(\pi) : C} \qquad \frac{\Gamma \vdash \pi : B}{\Gamma \vdash j(\pi) : C}\ \vee\text{-intro2, if } C \equiv A \vee B$$

$$\frac{\Gamma \vdash \pi_1 : D \qquad \Gamma, \alpha : A \vdash \pi_2 : C \qquad \Gamma, \beta : B \vdash \pi_3 : C}{\Gamma \vdash (\delta\ \pi_1\ \alpha\pi_2\ \beta\pi_3) : C}\ \vee\text{-elim, if } D \equiv A \vee B$$

$$\frac{\Gamma \vdash \pi : B}{\Gamma \vdash (\delta_\perp\ \pi) : A}\ \perp\text{-elim, if } B \equiv \perp$$

$$\frac{\Gamma \vdash \pi : A}{\Gamma \vdash \lambda x.\pi : B}\ \forall\text{-intro, if } B \equiv \forall x A, \text{ and } x \notin FV(\Gamma)$$

$$\frac{\Gamma \vdash \pi : B}{\Gamma \vdash (\pi\ t) : C}\ \forall\text{-elim if } B \equiv \forall x A, \text{ and } C \equiv (t/x)A$$

$$\frac{\Gamma \vdash \pi : C}{\Gamma \vdash \langle t, \pi \rangle : B}\ \exists\text{-intro if } B \equiv \exists x A, \text{ and } C \equiv (t/x)A$$

$$\frac{\Gamma \vdash \pi : C \qquad \Gamma, \alpha : A \vdash \pi' : B}{\Gamma \vdash (\delta_\exists\ \pi\ x\alpha\pi') : B}\ \exists\text{-elim, if } C \equiv \exists x A, \text{ and } x \notin FV(\Gamma, B)$$

Figure 1 Intuitionistic natural deduction modulo.

Each proof-term construction corresponds to a natural deduction rule: terms of the form α express proofs built with the axiom rule, terms of the form $\lambda\alpha\ \pi$ and $(\pi\ \pi')$ express proofs built respectively with the introduction and elimination rules of the implication,

terms of the form $\langle \pi, \pi' \rangle$ and $fst(\pi)$, $snd(\pi)$ express proofs built with the introduction and elimination rules of the conjunction, terms of the form $i(\pi), j(\pi)$ and $(\delta\ \pi_1\ \alpha\pi_2\ \beta\pi_3)$ express proofs built with the introduction and elimination rules of the disjunction, terms of the form $(\delta_\perp\ \pi)$ express proofs built with the elimination rule of the contradiction, terms of the form $\lambda x\ \pi$ and $(\pi\ t)$ express proofs built with the introduction and elimination rules of the universal quantifier and terms of the form $\langle t, \pi \rangle$ and $(\delta_\exists\ \pi\ x\alpha\pi')$ express proofs built with the introduction and elimination rules of the existential quantifier.
We call *neutral* those proof-terms that are formed with an elimination rule or an axiom.

For example, in predicate logic with two 0-ary predicates P and Q, and in a theory defined by a congruence relation $\equiv$ such that $P \equiv (Q \Rightarrow Q)$, the proof-term $\lambda\alpha.\alpha$ is a proof of P in the empty context, with the rules $\Rightarrow$-intro and axiom as proof derivation.

Capture avoiding substitution in proof-terms is defined as usual. Notice that both term-variables and proof-variables can be substituted respectively by terms and proof-terms. The substitution of the variable x (resp. proof-variable α) by the term t (resp. proof-term π') in the proof-term π is written $(t/x)\pi$ (resp. $(\pi'/\alpha)\pi$).

Cut elimination in proof derivations is done, via the Curry-Howard correspondence, by β-reduction on proof-terms. β-reduction is defined as the smallest contextual closure of the following reduction rules (corresponding, respectively, to the elimination of a cut $\Rightarrow$-e/$\Rightarrow$-i, $\wedge$-e1/$\wedge$-i, $\wedge$-e2/$\wedge$-i, $\vee$-e/$\vee$-i1, $\vee$-e/$\vee$-i2, $\forall$-e/$\forall$-i and $\exists$-e/$\exists$-i).

$$\begin{array}{rcl}
(\lambda\alpha.\pi_1\ \pi_2) & \rhd & (\pi_2/\alpha)\pi_1 \\
fst(\langle \pi_1, \pi_2 \rangle) & \rhd & \pi_1 \\
snd(\langle \pi_1, \pi_2 \rangle) & \rhd & \pi_2 \\
(\delta\ i(\pi_1)\ \alpha\pi_2\ \beta\pi_3) & \rhd & (\pi_1/\alpha)\pi_2 \\
(\delta\ j(\pi_1)\ \alpha\pi_2\ \beta\pi_3) & \rhd & (\pi_1/\beta)\pi_3 \\
\lambda x.\pi\ t & \rhd & (t/x)\pi \\
(\delta_\exists\ \langle t, \pi_1 \rangle\ \alpha x\pi_2) & \rhd & (t/x, \pi_1/\alpha)\pi_2
\end{array}$$

Figure 2 Proof-term reduction rules.

Let π be a proof-term. We write $\pi \rhd^* \pi'$ if π β-reduces to π' in zero or more steps. π is said in normal form if no reduction rule applies to π. It is weakly (resp. strongly) normalizing if there exists a finite β-reduction sequence $\pi \rhd^* \pi'$ with π' in normal form (resp. all β-reduction sequences from π are finite). By extension, a theory is weakly (resp. strongly) normalizing if all proof-terms that are proofs of formulæ of that theory, are weakly (resp. strongly) normalizing. Since a step of cut elimination in a proof derivation corresponds to a β-reduction step of the corresponding proof-term, one can prove syntactically that all weakly normalizing theories satisfy the cut elimination property.

Finally, notice that β-reduction is confluent for all theories expressed in deduction modulo, i.e. if π, π_1, π_2 are proof-terms such that $\pi \rhd^* \pi_1$ and $\pi \rhd^* \pi_2$ then there exists a proof-term π' such that $\pi_1 \rhd^* \pi'$ and $\pi_2 \rhd^* \pi'$.

3 Pre-Heyting algebras and pre-models

In [13], Dowek and Werner have generalized the Tait-Girard's reducibility method, by defining the notion of reducibility candidates for deduction modulo, namely *pre-models*, whose existence is a sufficient condition for strong normalization. Later, Dowek exhibited the notion of *pre-Heyting algebras* [9] (also known as pseudo-Heyting algebras or Truth Values Algebras) which is the underlying structure of those pre-models. Let us first recall the definitions of those pre-Heyting algebras and of models valued on them.

3.1 (pre-) Heyting algebras

▶ **Definition 1** (pre-Heyting algebra). Let $\mathcal{B}$ be a set, $\leq$ be a relation on $\mathcal{B}$, $\mathcal{A}$ and $\mathcal{E}$ be subsets of $\wp(\mathcal{B})$, $\tilde{\top}$, $\tilde{\bot}$ be elements of $\mathcal{B}$, $\tilde{\Rightarrow}$, $\tilde{\wedge}$, and $\tilde{\vee}$ be functions from $\mathcal{B} \times \mathcal{B}$ to $\mathcal{B}$, $\tilde{\forall}$ be a function from $\mathcal{A}$ to $\mathcal{B}$ and $\tilde{\exists}$ be a function from $\mathcal{E}$ to $\mathcal{B}$.
The structure $\mathcal{B} = \langle \mathcal{B}, \leq, \mathcal{A}, \mathcal{E}, \tilde{\top}, \tilde{\bot}, \tilde{\Rightarrow}, \tilde{\wedge}, \tilde{\vee}, \tilde{\forall}, \tilde{\exists} \rangle$ is said to be a *pre-Heyting algebra* if

- the relation $\leq$ is a pre-order,
- $\tilde{\bot}$ is a minimum element,
- $\tilde{\top}$ is a maximum element,
- for all a, b in $\mathcal{B}$, $a \mathrel{\tilde{\wedge}} b$ is a greatest lower bound of a and b
  and $a \mathrel{\tilde{\vee}} b$ is a least upper bound of a and b,
- $\tilde{\forall}$ and $\tilde{\exists}$ are infinite greatest lower bound and least upper bound, respectively,
- for all a, b, c in $\mathcal{B}$, $a \leq b \mathrel{\tilde{\Rightarrow}} c$ if and only if $a \mathrel{\tilde{\wedge}} b \leq c$.

Compared to [9], we drop the closure conditions that $a \mathrel{\tilde{\Rightarrow}} A$ and $E \mathrel{\tilde{\Rightarrow}} a$ are both in $\mathcal{A}$. This simplification is possible because we do not reason within Truth Values Algebras. See [9] for more detailed definitions and explanations.

▶ **Definition 2** (Heyting algebra). A pre-Heyting algebra is said to be a *Heyting algebra* if the pre-order $\leq$ is antisymmetric and therefore an order.

3.2 Models

Let us define now the notion of model valued on a pre-Heyting algebra.

▶ **Definition 3** ($\mathcal{B}$-valued structure).
Let $\mathcal{L} = \langle f_i, P_j \rangle$ be a language in first order logic and $\mathcal{B}$ be a pre-Heyting algebra, a $\mathcal{B}$-*valued structure* for the language $\mathcal{L}$, $\mathcal{M} = \langle M, \mathcal{B}, \hat{f}_i, \hat{P}_j \rangle$ is a structure such that $\hat{f}_i$ is a function from M^n to M where n is the arity of the function symbol f_i and $\hat{P}_j$ is a function from M^n to $\mathcal{B}$ where n is the arity of the predicate symbol P_j.

▶ **Definition 4** (Environments).
Given a $\mathcal{B}$-valued structure $\mathcal{M} = \langle M, \mathcal{B}, \hat{f}_i, \hat{P}_j \rangle$, an *environment* is a function which associates an element of M with each term variable.

▶ **Definition 5** (Denotation). Let $\mathcal{B}$ be a pre-Heyting algebra, $\mathcal{M}$ a $\mathcal{B}$-valued structure and ϕ an environment. The denotation $[\![A]\!]^{\mathcal{M}}_{\phi}$ of a formula A in $\mathcal{M}$ is inductively defined from $\mathcal{B}$ and ϕ as follows:

- $[\![x]\!]^{\mathcal{M}}_{\phi} = \phi(x)$,
- $[\![f(t_1, ..., t_n)]\!]^{\mathcal{M}}_{\phi} = \hat{f}([\![t_1]\!]^{\mathcal{M}}_{\phi}, ..., [\![t_n]\!]^{\mathcal{M}}_{\phi})$,
- $[\![P(t_1, ..., t_n)]\!]^{\mathcal{M}}_{\phi} = \hat{P}([\![t_1]\!]^{\mathcal{M}}_{\phi}, ..., [\![t_n]\!]^{\mathcal{M}}_{\phi})$,
- $[\![\bot]\!]^{\mathcal{M}}_{\phi} = \tilde{\bot}$,
- $[\![A \Rightarrow B]\!]^{\mathcal{M}}_{\phi} = [\![A]\!]^{\mathcal{M}}_{\phi} \mathrel{\tilde{\Rightarrow}} [\![B]\!]^{\mathcal{M}}_{\phi}$,
- $[\![A \wedge B]\!]^{\mathcal{M}}_{\phi} = [\![A]\!]^{\mathcal{M}}_{\phi} \mathrel{\tilde{\wedge}} [\![B]\!]^{\mathcal{M}}_{\phi}$,
- $[\![A \vee B]\!]^{\mathcal{M}}_{\phi} = [\![A]\!]^{\mathcal{M}}_{\phi} \mathrel{\tilde{\vee}} [\![B]\!]^{\mathcal{M}}_{\phi}$,

- $[\![\forall x\ A]\!]^{\mathcal{M}}_{\phi} = \tilde{\forall}\ \{[\![A]\!]^{\mathcal{M}}_{\phi+\langle x,e\rangle} \mid e \in M\}$ when it is defined,
- $[\![\exists x\ A]\!]^{\mathcal{M}}_{\phi} = \tilde{\exists}\ \{[\![A]\!]^{\mathcal{M}}_{\phi+\langle x,e\rangle} \mid e \in M\}$ when it is defined.

▶ Remark. We omit $\mathcal{M}$ from $[\![A]\!]^{\mathcal{M}}_{\phi}$ when it is clear from context.
In all the pre-Heyting Algebras we consider in this paper, $\mathcal{A}$ and $\mathcal{E}$ at least contain all the sets of the form $\{[\![A]\!]_{\phi+\langle x,e\rangle} \mid e \in M\}$ so that $[\![A]\!]_{\phi}$ is always defined.
For any formula A, terms t, u and environment ϕ, we have $[\![(t/x)A]\!]_{\phi} = [\![A]\!]_{\phi+\langle x,[\![t]\!]_{\phi}\rangle}$ and $[\![(t/x)u]\!]_{\phi} = [\![u]\!]_{\phi+\langle x,[\![t]\!]_{\phi}\rangle}$

▶ **Definition 6** (Model). The $\mathcal{B}$-valued structure $\mathcal{M}$ is said to be *a model of* a theory $\mathcal{L}, \equiv$ if for all formulæ A and B and terms t and u such that $A \equiv B$ and $t \equiv u$, and for all environments ϕ, we have $[\![A]\!]_{\phi} = [\![B]\!]_{\phi}$ and $[\![t]\!]_{\phi} = [\![u]\!]_{\phi}$.

3.3 Reducibility candidates and pre-models for weak normalization

The main idea of Tait-Girard's reducibility method is to associate with each proposition A, a set R_A of strongly normalizing proof-terms, that verifies some *reducibility* conditions, and then show the so-called *adequacy lemma*: all proof-terms that are proofs of some formula A belong to R_A and are therefore strongly normalizing. We adapt here this notion of reducibility candidates to weak normalization. We shall write *WN-reducibility candidates.* We also write "*WN*" the set of weakly normalizing proof-terms and "π is *WN*" when $\pi \in WN$. A proof-term is said to be *neutral* if it is of the form $\alpha, (\pi\ \pi'), fst(\pi), snd(\pi), (\delta\ \pi_1\ \alpha\pi_2\ \beta\pi_3), (\pi\ t)$ or $(\delta_{\exists}\ \pi_1\ \alpha x\pi_2)$. A neutral proof-term is said to be *isolated* if all its reduction sequences end with a neutral proof-term (those proof-terms are called *hereditarily neutral* in [19]).

▶ **Definition 7** (Reducibility candidates for weak normalization).
A set R of proof-terms is a *WN-reducibility candidate* if and only if:
$(\mathbf{P}_1)$ if $\pi \in R$, then π is weakly normalizing,
$(\mathbf{P}_{3a})$ if π is neutral and there exists $\pi' \in R$ such that $\pi \rhd \pi'$, then $\pi \in R$.
$(\mathbf{P}_{3b})$ R contains all isolated weakly normalizing proof-terms.

If we compare this definition to usual reducibility candidates (see [16]), $(\mathbf{CR}_1)$ becomes $(\mathbf{P}_1)$, stability by reduction $(\mathbf{CR}_2)$ is not needed for our particular purpose of proving, via semantic methods, the cut elimination property (albeit we shall see in Section 5.1 that it could be useful to impose this property), and we split the usual property $(\mathbf{CR}_3)$ of reducibility candidates into $(\mathbf{P}_{3a})$ and $(\mathbf{P}_{3b})$: $(\mathbf{P}_{3a})$ is the adaptation of $(\mathbf{CR}_3)$ to weak normalization but this no longer entails the non-emptiness of the considered set (a crucial point in the proof of the adequacy lemma). $(\mathbf{P}_{3b})$ ensures that non-emptiness.

Let us now define the pre-Heyting algebra of *WN*-reducibility candidates.

▶ **Definition 8** (Operations).
If E and F are sets of proof-terms, and $\mathcal{F}$ is a set of sets of proof-terms,

- The sets $\tilde{\top}$ and $\tilde{\bot}$ are both the set of weakly normalizing proof-terms.
- $E\ \tilde{\Rightarrow}\ F$ is the set of proof-terms π such that either π is isolated and weakly normalizing, or there exists π_1 such that $\pi \rhd^* \lambda\alpha\ \pi_1$ and for all $\pi' \in E$, $(\pi'/\alpha)\pi_1 \in F$.
- $E\ \tilde{\wedge}\ F$ is the set of proof-terms π such that either π is isolated and weakly normalizing, or there exists π_1, π_2 such that $\pi \rhd^* \langle\pi_1, \pi_2\rangle$, $\pi_1 \in E$ and $\pi_2 \in F$.
- $E\ \tilde{\vee}\ F$ is the set of proof-terms π such that either π is isolated and weakly normalizing, or there exists π_1 such that $\pi \rhd^* i(\pi_1)$ (resp. $j(\pi_1)$) and $\pi_1 \in E$ (resp. F).

- $\tilde{\forall}\ \mathcal{F}$ is the set of proof-terms π such that either π is isolated and weakly normalizing, or there exists π_1 such that $\pi \rhd^* \lambda x\ \pi_1$ and for all terms t and $G \in \mathcal{F}$, $(t/x)\pi_1$ is in G.
- $\tilde{\exists}\ \mathcal{F}$ is the set of proof-terms π such that either π is isolated and weakly normalizing, or there exists π_1, $G \in \mathcal{F}$ and a term t such that $\pi \rhd^* \langle t, \pi_1\rangle$ and $\pi_1 \in G$.

▶ Remark. We could also have left $\tilde{\bot}$ be the smallest (for inclusion) reducibility candidate, *i.e.* the set of isolated weakly normalizing terms. But since we are going to choose the trivial pre-order, there is no particular reason to do so.

▶ **Lemma 9.** *The set of WN-reducibility candidates is closed by the operations of Def. 8.*

We can first remark that the set WN is a WN-reducibility candidate. Moreover, if E, F are WN-reducibility candidates and $\mathcal{F}$ is a set of WN-reducibility candidates, $E \tilde{\Rightarrow} F$, $E \tilde{\wedge} F$, $E \tilde{\vee} F$, $\tilde{\forall}\ \mathcal{F}$ and $\tilde{\exists}\ \mathcal{F}$ contains all isolated weakly-normalizing proof-terms by definition hence those sets satisfy $(\mathbf{P}_{3b})$.

$\tilde{\Rightarrow}$) $(\mathbf{P}_1)$ Let $\pi \in E \tilde{\Rightarrow} F$. Either π is isolated and weakly normalizing. Or there exists π_1 such that $\pi \rhd^* \lambda\alpha\ \pi_1$ and for all $\pi' \in E$, $(\pi'/\alpha)\pi_1 \in F$. Since E satisfies $(\mathbf{P}_{3b})$, we have $\alpha \in E$, hence $\pi_1 \in F \subseteq$WN, since F satisfies P_1 and so do $\lambda\alpha.\pi_1$ and finally π.

$(\mathbf{P}_{3a})$ Let π be a neutral proof-term and $\pi' \in E \tilde{\Rightarrow} F$ such that $\pi \rhd \pi'$. If π' is isolated and WN, then $\pi \in$WN and is also isolated, by confluence. Otherwise $\pi \rhd \pi' \rhd^* \lambda\alpha.\pi_1$ with $(\pi''/\alpha)\pi_1 \in F$ for all $\pi'' \in E$. In both cases, $\pi \in E \tilde{\Rightarrow} F$.

$\tilde{\wedge}$) $(\mathbf{P}_1)$ Let $\pi \in E \tilde{\wedge} F$. Either π is isolated and weakly normalizing. Or there exists π_1, π_2 such that $\pi \rhd^* \langle\pi_1, \pi_2\rangle$ with $\pi \in E \subseteq$WN and $\pi_2 \in F \subseteq$WN, hence $\pi \in$WN.

$(\mathbf{P}_{3a})$ Let π be a neutral proof-term and $\pi' \in E \tilde{\wedge} F$ such that $\pi \rhd \pi'$. If π' is isolated and WN, then $\pi \in$WN and is also isolated, by confluence. Otherwise $\pi \rhd \pi' \rhd^* \langle\pi_1, \pi_2\rangle$ with $\pi_1 \in E$ and $\pi_2 \in F$. In both cases, $\pi \in E \tilde{\wedge} F$.

$\tilde{\vee}$) $(\mathbf{P}_1)$ Let $\pi \in E \tilde{\vee} F$. Either π is isolated and weakly normalizing. Or there exists π_1 such that $\pi \rhd^* i(\pi_1)$ (resp. $j(\pi_1)$) with $\pi_1 \in E$ (resp. F). Since E and F satisfy $(\mathbf{P}_1)$, we have $\pi \in WN$.

$(\mathbf{P}_{3a})$ Let π be a neutral proof-term and $\pi' \in E \tilde{\vee} F$ such that $\pi \rhd \pi'$. If π' is isolated and WN, then $\pi \in$WN and is also isolated, by confluence. Otherwise $\pi \rhd \pi' \rhd^* i(\pi_1)$ (resp. $j(\pi_1)$) with $\pi_1 \in E$ (resp. F). In both cases, $\pi \in E \tilde{\vee} F$.

$\tilde{\forall}$) $(\mathbf{P}_1)$ Let $\pi \in \tilde{\forall}\mathcal{F}$. Either π is isolated and weakly normalizing. Or there exists π_1 such that $\pi \rhd^* \lambda x\ \pi_1$ and for all terms t and sets $G \in \mathcal{F}$, $(t/x)\pi_1 \in \mathcal{F}$. Since each $G \in \mathcal{F}$ satisfies $(\mathbf{P}_1)$, we have, in particular, $\pi_1 = (x/x)\pi_1 \in$WN, and so do $\lambda x.\pi_1$ and finally π.

$(\mathbf{P}_{3a})$ Let π be a neutral proof-term and $\pi' \in \tilde{\forall}\mathcal{F}$ such that $\pi \rhd \pi'$. If π' is isolated and WN, then $\pi \in$WN and is also isolated, by confluence. Otherwise $\pi \rhd \pi' \rhd^* \lambda x.\pi_1$ with $(t/x)\pi_1 \in G$ for all terms t and $G \in \mathcal{F}$. In both cases, $\pi \in \tilde{\forall}\mathcal{F}$.

$\tilde{\exists}$) $(\mathbf{P}_1)$ Let $\pi \in \tilde{\exists}\mathcal{F}$. Either π is isolated and weakly normalizing. Or there exists π_1, $G \in \mathcal{F}$ and a term t such that $\pi \rhd^* \langle t, \pi_1\rangle$ with $\pi_1 \in G$. Hence $\pi_1 \in$WN and so does π.

$(\mathbf{P}_{3a})$ Let π be a neutral proof-term and $\pi' \in \tilde{\exists}\mathcal{F}$ such that $\pi \rhd \pi'$. If π' is isolated and WN, then $\pi \in$WN and is also isolated, by confluence. Otherwise $\pi \rhd \pi' \rhd^* \langle t, \pi_1\rangle$ with π_1 in some $G \in \mathcal{F}$. In both cases, $\pi \in \tilde{\exists}\mathcal{F}$.

▶ **Definition 10** (The algebra of WN-reducibility candidates).
The set $\mathcal{B}$ is the set of WN-reducibility candidates. The sets $\mathcal{A}$ and $\mathcal{E}$ are $\wp(\mathcal{B})$ (the powerset of $\mathcal{B}$). The operations are those of Def. 8. The pre-order is the trivial pre-order, *i.e.* $a \leq b$ for any $a, b \in \mathcal{B}$.

Notice that the existence of a model valued on this pre-Heyting algebra for some theory can be obtained by super-consistency [9] since this pre-Heyting algebra is full, ordered (using the inclusion) and complete.

Syntactic proof of (weak normalization and) cut elimination

In the following, we show how to adapt the usual reducibility method ([13] and [7, 8] for deduction modulo) to prove that the existence of a model valued on the pre-Heyting algebra of WN-reducbility candidates entails weak normalization (and therefore, syntactically, cut elimination) for the considered theory in deduction modulo.
We suppose that the denotation $[\![.]\!]_.$ of a theory forms a model valued on the pre-Heyting algebra of WN-reducibility candidates in the sense of Def. 6.

As usual, we work with proof-terms substitutions adapted to a typing context, that we call *assignments*.

▶ **Definition 11** (Assignments on a typing context). Given a typing context Γ and an environment ϕ, an assignment is a proof-term substitution σ such that for all declarations $\alpha : A$ in Γ, we have $\sigma\alpha \in [\![A]\!]_\phi^{\mathcal{M}}$. When it is clear from context, we may not precise the typing context on which some assignment is defined.

▶ **Lemma 12** (Adequacy lemma).
For all typing contexts Γ, formulæ A, environments ϕ, term substitutions θ, proof-terms π, and assignments σ on Γ,

$$\textit{if } \Gamma \vdash \pi : A \textit{ then } \sigma\theta\pi \in [\![A]\!]_\phi.$$

Proof. By induction on the length of the derivation of $\Gamma \vdash \pi : A$. By case analysis on the last rule.

- If the last rule is `axiom`, then π is a variable α and there exists some formula $B \equiv A$ and a declaration $\alpha : B$ in Γ. By hypothesis on the assignment, $\sigma\theta\alpha = \sigma\alpha \in [\![B]\!]_\phi = [\![A]\!]_\phi$ since $[\![.]\!]_.$ is a model.
- If the last rule is `⇒-intro`, then π is an abstraction $\lambda\alpha.\nu$ (we can suppose that α is not in the domain of σ by α-renaming), and there exists formulæ B and C such that $A \equiv B \Rightarrow C$ and $\Gamma, \alpha : B \vdash \nu : C$ (with a smaller derivation). For all $\mu \in [\![B]\!]_\phi$, $(\mu/\alpha)\sigma$ is an assignment on $\Gamma, \alpha : B$ hence by induction hypothesis $(\mu/\alpha)\theta\sigma\nu \in [\![C]\!]_\phi$. Finally, by Def. 8, $\sigma\theta\pi = \lambda\alpha.\sigma\theta\nu \in [\![B]\!]_\phi \tilde{\Rightarrow} [\![C]\!]_\phi = [\![B \Rightarrow C]\!]_\phi = [\![A]\!]_\phi$.
- If the last rule is `⇒-elim`, then π is an application $(\mu\ \nu)$, and there exists formulæ B and C such that $\Gamma \vdash \mu : C$, $\Gamma \vdash \nu : B$ (with smaller derivations), and $C \equiv B \Rightarrow A$. By induction hypothesis, we have $\sigma\theta\mu \in [\![C]\!]_\phi = [\![B \Rightarrow A]\!]_\phi = [\![B]\!]_\phi \tilde{\Rightarrow} [\![A]\!]_\phi$ and $\sigma\theta\nu \in [\![B]\!]_\phi$. Either $\sigma\theta\mu$ is isolated and weakly normalizing (since it belongs to $[\![C]\!]_\phi$), then so is $\sigma\theta\pi = \sigma\theta(\mu\ \nu) = (\sigma\theta\mu\ \sigma\theta\nu)$ which therefore belongs to $[\![A]\!]_\phi$ by $(\mathbf{P}_{3b})$. Or there exists α and π_1 such that $\sigma\theta\mu \rhd^* \lambda\alpha.\pi_1$ and $(\sigma\theta\nu/\alpha)\pi_1 \in [\![A]\!]_\phi$ since $\sigma\theta\mu \in [\![B]\!]_\phi \tilde{\Rightarrow} [\![A]\!]_\phi$. Finally $\sigma\theta\pi = (\sigma\theta\mu\ \sigma\theta\nu) \in [\![A]\!]_\phi$ since it is neutral and it reduces to $(\sigma\theta\nu/\alpha)\pi_1 \in [\![A]\!]_\phi$ (by a repeated use of $(\mathbf{P}_{3a})$).
- Proofs of the other cases follow the same scheme.

◀

In the following, we shall bypass that method and make appear an underlying Heyting algebra structure from WN-reducibility candidates in order to provide a full semantic proof that the existence of a model valued on WN-reducibility candidates entails cut elimination.

4 A Heyting algebra

Pre-Heyting Algebras can easily be turned into Heyting Algebras [9] by a quotient operation, but this is of no help here since with the algebra of Def. 10 we obtain a one-point (*i.e.* trivial) Heyting Algebra. In order to achieve our purpose and extract a non-trivial Heyting algebra from our structure of pre-models for weak normalization, we interpret propositions by sets of contexts (*outer values*) given by this notion of pre-model and we show that it forms a Heyting Algebra with well chosen operations.
For the following, we consider a theory in deduction modulo and we suppose that there exists a model $\mathcal{M}$, valued on the pre-Heyting algebra of *WN*-reducibility candidates, for that theory. We write $[\![.]\!]^{\mathcal{M}}$ for the denotation it defines.

4.1 Outer value

Notice that the algebra we are to build contains sets of contexts (i.e. sequences of unlabeled formulæ) and not typing contexts. We say that a typing context Δ is a *labeling* of a context $\Gamma = A_1, \dots, A_n$ if there exists proof-variables $\alpha_1, \dots, \alpha_n$ such that $\Delta = \alpha_1 : A_1, \dots, \alpha_n : A_n$.

▶ **Definition 13** (Outer Value). Let A be a formula. We define its *weak outer value* $\lfloor A \rfloor$ as the set of contexts Γ such that there exists a labeling Δ of Γ and a proof-term π with:

- $\Delta \vdash \pi : A$
- for any environment ϕ, any term-substitution θ, any assignment σ on Δ, $\sigma\theta\pi \in [\![A]\!]^{\mathcal{M}}_{\phi}$

▶ **Lemma 14.** *For all formulæ A, B and contexts Γ, we have*

- $A \in \lfloor A \rfloor$
- $\Gamma \in \lfloor A \rfloor$ *implies* $\Gamma, B \in \lfloor A \rfloor$
- $\lfloor A \rfloor = \lfloor B \rfloor$ *if* $A \equiv B$.

Proof. Each point is treated separately and is straightforward.

- Because the labeling $\alpha : A$ and the proof $\alpha : A \vdash \alpha : A$ verify Def. 13.
- If Δ is a labeling of Γ, π is a suitable proof-term and β is a fresh proof variable, the typing context $\Gamma, \beta : B$ and the proof $\Gamma, \beta : B \vdash \pi : A$ verify Def. 13.
- From Def. 6, the same proof-terms are $[\![A]\!]^{\mathcal{M}}$ and $[\![B]\!]^{\mathcal{M}}$. Moreover, if $\Gamma \vdash \pi : A$ then $\Gamma \vdash \pi : B$ is an easy property of Deduction modulo [13].

◀

In the following, for all (unlabeled) contexts Γ, proof-terms π and formulæ A, we shall write $\Gamma \vdash \pi : A$ if there exists a labeling Δ of Γ such that $\Delta \vdash \pi : A$.

4.2 The algebra

With the help of Def. 13 we can now define a Heyting algebra:

▶ **Definition 15** (Heyting algebra Ω).
We define Ω to be the set containing all the $\lfloor A \rfloor$ for any formula A. It is ordered by inclusion. $\mathcal{A}$ and $\mathcal{E}$ are both equal to the set of all $\{\lfloor (t/x)A \rfloor \mid t \in \mathcal{T}\}$ for any formula A, where $\mathcal{T}$ is the set of open terms. We define the operations and constants to be:

- $\check{\bot} = \lfloor \bot \rfloor$
- $\check{\top} = \lfloor \bot \Rightarrow \bot \rfloor$
- $\lfloor A \rfloor \mathbin{\check{\wedge}} \lfloor B \rfloor = \lfloor A \wedge B \rfloor$
- $\lfloor A \rfloor \mathbin{\check{\vee}} \lfloor B \rfloor = \lfloor A \vee B \rfloor$

- $\lfloor A \rfloor \check{\Rightarrow} \lfloor B \rfloor = \lfloor A \Rightarrow B \rfloor$
- $\check{\forall} \{ \lfloor (t/x)A \rfloor \mid t \in \mathcal{T} \} = \lfloor \forall x A \rfloor$
- $\check{\exists} \{ \lfloor (t/x)A \rfloor \mid t \in \mathcal{T} \} = \lfloor \exists x A \rfloor$

▶ Remark. This definition should formally appear below Lem. 16 and 17, and not just above. Indeed, those lemmata ensure that the introduced operators are well-defined, and do not depend on them. However, we believe that presenting first Def. 15 is more natural.

To show that Def. 15 defines a Heyting algebra, we first prove that $\check{\wedge}$ is set intersection.

▶ **Lemma 16.** *For any formulæ A and B, $\lfloor A \rfloor \check{\wedge} \lfloor B \rfloor = \lfloor A \rfloor \cap \lfloor B \rfloor$.*

Proof.

- $\lfloor A \rfloor \cap \lfloor B \rfloor \subseteq \lfloor A \wedge B \rfloor$: Let $\Gamma \in \lfloor A \rfloor \cap \lfloor B \rfloor$. Let π_1 and π_2 be proof-terms verifying the conditions of Def. 13. Since $\Gamma \vdash \pi_1 : A$ and $\Gamma \vdash \pi_2 : B$, we have $\Gamma \vdash \langle \pi_1, \pi_2 \rangle : A \wedge B$. We claim that $\langle \pi_1, \pi_2 \rangle$ is a suitable proof-term that verifies the conditions of Def. 13: let ϕ be an environment, σ be an assignment and θ be a term-substitution. Then $\sigma\theta\langle \pi_1, \pi_2 \rangle = \langle \sigma\theta\pi_1, \sigma\theta\pi_2 \rangle$ and, since by Def. 13 $\sigma\theta\pi_1 \in [\![A]\!]^{\mathcal{M}}_{\phi}$ and $\sigma\theta\pi_2 \in [\![B]\!]^{\mathcal{M}}_{\phi}$, we get by Def. 8 that $\langle \sigma\theta\pi_1, \sigma\theta\pi_2 \rangle \in [\![A \wedge B]\!]^{\mathcal{M}}_{\phi}$.
- $\lfloor A \wedge B \rfloor \subseteq \lfloor A \rfloor \cap \lfloor B \rfloor$: we show only $\lfloor A \wedge B \rfloor \subseteq \lfloor A \rfloor$ since the other inclusion has exactly the same proof. Let $\Gamma \in \lfloor A \wedge B \rfloor$ and let π be a proof-term that verifies the conditions of Def. 13. Then $\Gamma \vdash fst(\pi) : A$ and we claim that $fst(\pi)$ is a suitable proof-term: let ϕ be an environment, σ be an assignment and θ be a term-substitution. Then $\sigma\theta fst(\pi) = fst(\sigma\theta\pi)$. By hypothesis, $\sigma\theta\pi \in [\![A \wedge B]\!]^{\mathcal{M}}_{\phi}$ and according to Def. 8 we have two choices. If $\sigma\theta\pi$ is *WN* isolated, then so is $fst(\sigma\theta\pi)$ and then $fst(\sigma\theta\pi) \in \lfloor A \rfloor$ by $(\mathbf{P}_{3b})$. Otherwise, $\sigma\theta\pi \rhd^* \langle \pi_1, \pi_2 \rangle$ ($\sigma\theta\pi$ cannot reduce to a non neutral term of another form since the theory is not confusing and the types of β-equivalent proof-terms are equivalent). with $\pi_1 \in [\![A]\!]^{\mathcal{M}}_{\phi}$ and $\pi_2 \in [\![B]\!]^{\mathcal{M}}_{\phi}$. Then we have the following sequence:

$$fst(\sigma\theta\pi) \rhd^* fst(\langle \pi_1, \pi_2 \rangle) \rhd^1 \pi_1$$

  Since every term but the last in this reduction sequence is neutral, we conclude by a repeated use of $(\mathbf{P}_{3a})$ that $fst(\sigma\theta\pi) \in [\![A]\!]^{\mathcal{M}}_{\phi}$.

◀

We check now that Def. 15 does define a Heyting algebra (since $\mathcal{M}$ is a model valued on the pre-Heyting algebra of *WN*-reducibility candidates).

▶ **Lemma 17.** *The constants $\check{\bot}$, $\check{\top}$ and the operators $\check{\wedge}$, $\check{\vee}$, $\check{\Rightarrow}$, $\check{\forall}$, $\check{\exists}$ are operators of a Heyting algebra, the order being inclusion.*

Proof. We check one by one each operator:

- $\check{\top}$ is the greatest element. Let $\lfloor C \rfloor \in \Omega$ and $\Gamma \in \lfloor C \rfloor$. Then $\Gamma \vdash \lambda\alpha.\alpha : \bot \Rightarrow \bot$ and to show $\Gamma \in \lfloor \bot \Rightarrow \bot \rfloor$ we claim that $\lambda\alpha.\alpha$ is a suitable proof-term that verifies the conditions of Def. 13. Let ϕ be an environment, σ be an assignment and θ be a term-substitution. $\sigma\theta(\lambda\alpha.\alpha) = \lambda\alpha.\alpha$ and $\lambda\alpha.\alpha \in [\![\bot \Rightarrow \bot]\!]^{\mathcal{M}}_{\phi}$ by Def. 8 since for any $\pi' \in [\![\bot]\!]^{\mathcal{M}}_{\phi}$, $(\pi'/\alpha)\alpha = \pi' \in [\![\bot]\!]^{\mathcal{M}}_{\phi}$.
- $\check{\bot}$ is the least element. Let $\lfloor A \rfloor \in \Omega$, let $\Gamma \in \lfloor \bot \rfloor$ and π be a proof term verifying the conditions of Def. 13. Then $\Gamma \vdash (\delta_{\bot}\ \pi) : A$ and we claim that $(\delta_{\bot}\ \pi)$ is a suitable proof-term (Def. 13). Let ϕ be an environment, σ be an assignment and θ be a term-substitution. $\sigma\theta(\delta_{\bot}\ \pi) = \delta_{\bot}\ \sigma\theta\pi$ is isolated since there is no $\delta_{\bot}$ reduction rule and it is *WN* since $\sigma\theta\pi$ is *WN* by hypothesis on π. So $\Gamma \in \lfloor A \rfloor$.

- $\lfloor A \wedge B \rfloor$ is the greatest lower bound of $\lfloor A \rfloor$ and $\lfloor B \rfloor$ follows directly from Lemma 16 and the fact that set intersection is the greatest lower bound for set inclusion.
- $\lfloor A \vee B \rfloor$ is the least upper bound of $\lfloor A \rfloor$ and $\lfloor B \rfloor$:
  - $\lfloor A \rfloor \subseteq \lfloor A \vee B \rfloor$. Let $\Gamma \in \lfloor A \rfloor$ and π a proof-term verifying the conditions of Def. 13. Then $\Gamma \vdash i(\pi) : A \vee B$ and we claim that $i(\pi)$ is a suitable proof-term. Let ϕ be an environment, σ be an assignment and θ be a term-substitution. $\sigma\theta i(\pi) = i(\sigma\theta\pi)$ and we know by hypothesis that $\sigma\theta\pi \in [\![A]\!]^{\mathcal{M}}_{\phi}$. By Def. 8, $i(\sigma\theta\pi) \in [\![A \vee B]\!]^{\mathcal{M}}_{\phi}$.
  - $\lfloor B \rfloor \subseteq \lfloor A \vee B \rfloor$ has exactly the same proof.
  - if $\lfloor A \rfloor \subseteq \lfloor C \rfloor$ and $\lfloor B \rfloor \subseteq \lfloor C \rfloor$ then $\lfloor A \vee B \rfloor \subseteq \lfloor C \rfloor$. Let C be a formula such that $\lfloor A \rfloor \subseteq \lfloor C \rfloor$ and $\lfloor B \rfloor \subseteq \lfloor C \rfloor$, let $\Gamma \in \lfloor A \rfloor$ and let π be the proof-term verifying the conditions of Def. 13 such that $\Gamma \vdash \pi : A \vee B$. We show that $\Gamma \in \lfloor C \rfloor$.
    By Lem. 14 $A \in \lfloor A \rfloor \subseteq \lfloor C \rfloor$ and $\Gamma, A \in \lfloor C \rfloor$. Let π_1 be the proof-term verifying the conditions of Def. 13 such that $\Gamma, \alpha : A \vdash \pi_1 : C$. Analogously let π_2 be the suitable proof-term such that $\Gamma, \beta : B \vdash \pi_2 : C$. We build the proof:
    $$\dfrac{\Gamma, \alpha : A \vdash \pi_1 : C \qquad \Gamma, \beta : B \vdash \pi_2 : C \qquad \Gamma \vdash \pi : A \vee B}{\Gamma \vdash (\delta\ \pi\ \alpha\pi_1\ \beta\pi_2) : C}\ \vee\text{-elim}$$
    We claim that $(\delta\ \pi\ \alpha\pi_1\ \beta\pi_2)$ is a suitable proof-term. Let ϕ be an environment, σ be an assignment and θ be a term-substitution. Up to an α-renaming of α, β we have $\sigma\theta(\delta\pi\ \alpha\pi_1\ \beta\pi_2) = (\delta\ \sigma\theta\pi\ \alpha\sigma\theta\pi_1\ \beta\sigma\theta\pi_2)$. $\sigma\theta\pi \in [\![A \vee B]\!]^{\mathcal{M}}_{\phi}$. If $\sigma\theta\pi$ is *WN* isolated, then so is $\sigma\theta(\delta\ \pi\ \alpha\pi_1\ \beta\pi_2)$ and this proof-term belongs to $[\![C]\!]^{\mathcal{M}}_{\phi}$ by $(\mathbf{P}_{3b})$. Otherwise, following Def. 8, assume that $\sigma\theta\pi \rhd^* i(\pi')$ with $\pi' \in [\![A]\!]^{\mathcal{M}}_{\phi}$ (the other case is similar). Then we have the reduction sequence:
    $$\sigma\theta(\delta\ \pi\ \alpha\pi_1\ \beta\pi_2) \rhd^* (\delta\ i(\pi')\ \alpha\sigma\theta\pi_1\ \beta\sigma\theta\pi_2) \rhd (\pi'/\alpha)\sigma\theta\pi_1$$
    Since $\pi' \in [\![A]\!]^{\mathcal{M}}_{\phi}$, $(\pi'/\alpha)\sigma = \sigma + (\pi'/\alpha)$ is an assignment on $\Gamma, \alpha : A$ and by hypothesis on π_1 $(\pi'/\alpha)\sigma\pi_1 \in [\![C]\!]^{\mathcal{M}}_{\phi}$. Every term but the last in the above reduction sequence is neutral, so we conclude by a repeated use of $(\mathbf{P}_{3a})$ that $\sigma\theta(\delta\ \pi\ \alpha\pi_1\ \beta\pi_2) \in [\![C]\!]^{\mathcal{M}}_{\phi}$.
- $\lfloor A \Rightarrow B \rfloor$ is an implication operator. From Def. 1 (see also [21]) we must check:
  $$\lfloor A \rfloor \leq \lfloor B \rfloor \check{\Rightarrow} \lfloor C \rfloor \text{ iff } \lfloor A \rfloor \check{\wedge} \lfloor B \rfloor \leq \lfloor C \rfloor$$
  - if part: by Lem. 16 we can assume $\lfloor A \rfloor \cap \lfloor B \rfloor \subseteq \lfloor C \rfloor$. Let $\Gamma \in \lfloor A \rfloor$. By Lem. 14, $\Gamma, B \in \lfloor A \rfloor \cap \lfloor B \rfloor$, so by hypothesis, $\Gamma, B \in \lfloor C \rfloor$. Let π be the proof-term verifying the conditions of Def. 13 such that $\Gamma, \alpha : B \vdash \pi : C$. Then $\Gamma \vdash \lambda\alpha.\pi : B \Rightarrow C$ and we claim that $\lambda\alpha.\pi$ is a suitable proof-term (Def. 13). Let ϕ be an environment, σ be an assignment, θ be a term-substitution and assume without loss of generality that α is fresh, so that $\sigma\theta\lambda\alpha.\pi = \lambda\alpha.\sigma\theta\pi$. Let $\pi' \in [\![B]\!]^{\mathcal{M}}_{\phi}$, then, by Def. 8, we must show that $(\pi'/\alpha)\sigma\theta\pi = (\sigma + (\pi'/\alpha))\theta\pi \in [\![C]\!]^{\mathcal{M}}_{\phi}$ but this is immediate by hypothesis on π.
  - only if part: by Lem. 16, let $\Gamma \in \lfloor A \rfloor \cap \lfloor B \rfloor$. By hypothesis, $\Gamma \in \lfloor B \rfloor \check{\Rightarrow} \lfloor C \rfloor = \lfloor B \Rightarrow C \rfloor$. Let π and π_B the proof-terms verifying the conditions of Def. 13 such that $\Gamma \vdash \pi : B \Rightarrow C$ and $\Gamma \vdash \pi_B : B$. We have $\Gamma \vdash (\pi\ \pi_B) : C$ and we claim that $(\pi\ \pi_B)$ is a suitable proof-term (Def. 13). Let ϕ be an environment, σ be an assignment and θ be a term-substitution. $\sigma\theta\pi \in [\![B \Rightarrow C]\!]^{\mathcal{M}}_{\phi}$ by hypothesis. If it is *WN* isolated, then so is $\sigma\theta(\pi\ \pi_B)$ and therefore it belongs to $[\![C]\!]^{\mathcal{M}}_{\phi}$. Otherwise, following Def. 8, assume that $\sigma\theta\pi \rhd^* \lambda\alpha\pi_1$ with π_1 verifying the associated hypothesis. Then we have the reduction sequence:
    $$\sigma\theta(\pi\ \pi_B) \rhd^* \lambda\alpha.\pi_1\ \sigma\theta\pi_B \rhd (\sigma\theta\pi_B/\alpha)\pi_1$$

By hypothesis on π_1 and since $\sigma\theta\pi_B \in [\![B]\!]^{\mathcal{M}}_{\phi}$, $(\sigma\theta\pi_B/\alpha)\pi_1 \in [\![C]\!]^{\mathcal{M}}_{\phi}$, and by a repeated use of $(\mathbf{P}_{3a})$, $\sigma\theta(\pi\ \pi_B) \in [\![C]\!]^{\mathcal{M}}_{\phi}$.

- $\lfloor \forall x A\rfloor$ is the greatest lower bound of the set $\{\lfloor (t/x)A\rfloor \mid t \in \mathcal{T}\}$:
  - if for any t, $\lfloor C\rfloor \subseteq \lfloor (t/x)A\rfloor$ then $\lfloor C\rfloor \subseteq \lfloor \forall x A\rfloor$. Let $\Gamma \in \lfloor C\rfloor$. In particular, assuming without loss of generality that x is fresh, $\Gamma \in \lfloor (x/x)A\rfloor$. Let π be the proof-term that verifies the conditions of Def. 13. Then $\Gamma \vdash \lambda x.\pi : \forall x A$, and we claim that $\lambda x.\pi$ is a suitable proof-term. Let ϕ be an environment, σ be an assignment and θ be a term-substitution. Assuming for simplicity that x does not appear in $\sigma\theta$, $\sigma\theta\lambda x.\pi = \lambda x.\sigma\theta\pi$, and we show that for any term t and any $d \in M$, $(t/x)\sigma\theta\pi \in [\![A]\!]^{\mathcal{M}}_{\phi+(d/x)}$. This is immediate by hypothesis on π, applied to the environment $\phi + (d/x)$, the assignment σ and the term-substitution $\theta + (t/x)$.
  - $\lfloor \forall x A\rfloor \subseteq \lfloor (t/x)A\rfloor$ for any t: let $\Gamma \in \lfloor \forall x A\rfloor$, and let π a proof-term that verifies the conditions of Def. 13. Then $\Gamma \vdash (\pi\ t) : (t/x)A$ and we claim that $(\pi\ t)$ is a suitable proof-term. Let ϕ be an environment, σ be an assignment and θ be a term-substitution. If $\sigma\theta\pi$ is *WN* isolated then so is $\sigma\theta(\pi\ t)$ and this proof-term belongs to $[\![(t/x)A]\!]^{\mathcal{M}}_{\phi}$. Otherwise, $\sigma\theta\pi \rhd^* \lambda x.\pi_1$ and $(u/x)\pi_1 \in [\![A]\!]^{\mathcal{M}}_{\phi+(d/x)}$ for any term u and any $d \in M$, in particular for θt and $[\![t]\!]^{\mathcal{M}}_{\phi}$. Thus, by remark 3.2, $(\theta t/x)\pi_1 \in [\![(t/x)A]\!]^{\mathcal{M}}_{\phi}$ and by a repeated use of $(\mathbf{P}_{3b})$, $\sigma\theta(\pi\ t) \in [\![(t/x)A]\!]^{\mathcal{M}}_{\phi}$.

- $\lfloor \exists x A\rfloor$ is the least upper bound of the set $\{\lfloor (t/x)A\rfloor \mid t \in \mathcal{T}\}$:
  - $\lfloor (t/x)A\rfloor \subseteq \lfloor \exists x A\rfloor$ for any t: let t be a term, $\Gamma \in \lfloor (t/x)A\rfloor$ and let π a proof-term that verifies the conditions of Def. 13. Then $\Gamma \vdash \langle t, \pi\rangle : \exists x A$ and we claim that $\langle t, \pi\rangle$ is a suitable proof-term (Def. 13). Let ϕ be an environment, σ be an assignment and θ be a term-substitution. $\sigma\theta\langle t, \pi\rangle$ respects the condition of Def. 8 since it is such that $\sigma\theta\pi \in [\![(t/x)A]\!]^{\mathcal{M}}_{\phi} = [\![A]\!]^{\mathcal{M}}_{\phi+[\![t]\!]^{\mathcal{M}}_{\phi}}$ by hypothesis and remark 3.2.
  - if, for any t, $\lfloor (t/x)A\rfloor \subseteq \lfloor C\rfloor$, then $\lfloor \exists x A\rfloor \subseteq \lfloor C\rfloor$: Let $\Gamma \in \lfloor \exists x A\rfloor$. In particular, assuming without loss of generality that x is fresh, $\lfloor (x/x)A\rfloor \subseteq \lfloor C\rfloor$, and by Lem. 14 we have $\Gamma, A \in \lfloor C\rfloor$. Let π and π' be the proof-terms verifying the conditions of Def. 13 such that $\Gamma \vdash \pi : \exists x A$ and $\Gamma, \alpha : A \vdash \pi' : C$, with α a fresh proof variable. We build the proof:

$$\frac{\Gamma \vdash \pi : \exists x A \qquad \Gamma, \alpha : A \vdash \pi' : C}{\Gamma \vdash (\delta_\exists\ \pi\ x\alpha\pi') : C}$$

We claim that $(\delta_\exists\ \pi\ x\alpha\pi')$ is a suitable proof-term (Def. 13). Let ϕ be an environment, σ be an assignment and θ be a term-substitution. If $\sigma\theta\pi$ is *WN* isolated, then so is $\sigma\theta(\delta_\exists\ \pi\ x\alpha\pi')$ and this proof-term belongs to $[\![C]\!]^{\mathcal{M}}_{\phi}$. Otherwise $\sigma\theta\pi \rhd^* \langle t_1, \pi_1\rangle$ and $\pi_1 \in [\![A]\!]^{\mathcal{M}}_{\phi+(d/x)}$ for some $d \in M$. Assuming for simplicity that x and α do not appear in σ nor in θ:

$$\sigma\theta(\delta_\exists\ \pi\ x\alpha\pi') \rhd^* \delta_\exists\ \langle t_1, \pi_1\rangle\ x\alpha\pi' \rhd (t/x, \pi_1/\alpha)\sigma\theta\pi'$$

Since $\sigma + (\pi_1/\alpha)$ is a suitable assignment (adapted to the environment $\phi + (d/x)$), we have, by hypothesis on π, $(\sigma + (\pi_1/\alpha))(\theta + (t/x))\pi' \in [\![C]\!]^{\mathcal{M}}_{\phi+(d/x)} = [\![C]\!]^{\mathcal{M}}_{\phi}$ so, that by a repeated use of $(\mathbf{P}_{3a})$ we conclude that $\sigma\theta(\delta_\exists\ \pi\ x\alpha\pi') \in [\![C]\!]^{\mathcal{M}}_{\phi}$.

◀

We can conclude finally that Ω is a Heyting algebra when $\mathcal{M}$ is a model valued on the pre-Heyting algebra of WN-reducibility candidates[1]. We shall see now how it allows to prove the cut elimination property for the considered theory.

5 Model and cut elimination

We continue to suppose that we have a theory, defined by a language $\langle f_i, P_j\rangle$ and a congruence relation $\equiv$, that has a model valued on the pre-Heyting algebra of WN-reducibility candidates. We build a model valued on the Heyting algebra Ω of the previous section and we prove that the existence of such a model implies the cut elimination property.

▶ **Definition 18** (Heyting algebra model interpretation).
We define $\mathcal{D}$ as the Ω-valued structure $\langle \mathcal{T}', \Omega, \hat{f}_i, \hat{P}_j\rangle$ where:

- $\mathcal{T}'$ is the set of classes modulo $\equiv$ of open terms (the class of t is denoted $\overline{t}$),
- for any n-ary function symbol f: $\hat{f}^{\mathcal{D}}(\overline{t_1}, ..., \overline{t_n}) = \overline{f(t_1, ..., t_n)}$,
- for any n-ary predicate symbol P: $\hat{P}^{\mathcal{D}}(\overline{t_1}, ..., \overline{t_n}) = \lfloor P(t_1, ..., t_n)\rfloor$.

The last definition is well-formed since the function $t_1, ..., t_n \mapsto \lfloor P(t_1, ..., t_n)\rfloor$ is constant on classes of terms modulo $\equiv$.

As explained in the following lemma, $\lfloor . \rfloor$ is the denotation generated by Def. 18.

▶ **Lemma 19.** *For all terms t, formulæ A, assignments ϕ taking their values in $\mathcal{T}'$, and substitutions σ such that $\overline{\sigma(x)} = \phi(x)$ for any variable x, we have $[\![t]\!]^{\mathcal{D}}_{\phi} = \overline{\sigma t}$ and $[\![A]\!]^{\mathcal{D}}_{\phi} = \lfloor \sigma A\rfloor$.*

Proof. By structural induction on t and then A. There is a little subtlety in the case of quantifiers. Let us process the $\forall$ case, assuming that x is a fresh variable symbol:

$$\begin{array}{rcl} [\![\forall x\ A]\!]^{\mathcal{D}}_{\phi} & = & \tilde{\forall}\{[\![A]\!]^{\mathcal{D}}_{\phi+(\overline{t}/x)} \mid \overline{t} \in \mathcal{T}'\} \\ & = & \tilde{\forall}\{\lfloor(\sigma + (t/x))A\rfloor \mid \overline{t} \in \mathcal{T}'\} \quad = \quad \tilde{\forall}\{\lfloor(\sigma + (t/x))A\rfloor \mid t \in \mathcal{T}\} \\ & = & \lfloor \sigma \forall x\ A\rfloor \end{array}$$

The first equality is the definition of a model interpretation, the second comes by induction hypothesis, the last one is an application of Lem. 17, those steps are standard. Lem. 14 justifies the third equality on the second line: instead of having only one representative of the class $\overline{t}$, we can consider them all. ◀

▶ **Lemma 20.** *Let $t_1 \equiv t_2$ be two terms, $A \equiv B$ be two formulæ and ϕ be an assignment taking its values in $\mathcal{T}'$. Then $[\![t_1]\!]^{\mathcal{D}}_{\phi} = [\![t_2]\!]^{\mathcal{D}}_{\phi}$ and $[\![A]\!]^{\mathcal{D}}_{\phi} = [\![B]\!]^{\mathcal{D}}_{\phi}$.*

Proof. Let σ be a substitution fulfilling the hypothesis of Lem. 19. Then $\sigma t_1 \equiv \sigma t_2$, and $\overline{\sigma t_1} = \overline{\sigma t_2}$ and the result follows by Lem. 19. Similarly, $\sigma A \equiv \sigma B$, $\lfloor \sigma A\rfloor = \lfloor \sigma B\rfloor$ by Lem. 14 and the result follows by Lem. 19. ◀

Hence the model interpretation given by Def. 18 is adapted to the congruence $\equiv$. And we finally get the cut-elimination theorem:

▶ **Theorem 21.** *If the sequent $\Gamma \vdash A$ has a proof, then there exists a weakly normalizable proof of this sequent.*

[1] We can also show that it is not trivial ($\check{\top} \neq \check{\bot}$) but for this we need Thm. 21 that follows, to show, for instance, that the empty context does not belong to $\check{\bot}$

Proof. We let $[\![\Gamma]\!]^{\mathcal{D}} = \bigcap\{[\![A]\!]^{\mathcal{D}} \mid A \in \Gamma\}$ and $\lfloor\Gamma\rfloor = \bigcap\{\lfloor A\rfloor \mid A \in \Gamma\}$. By usual soundness w.r.t. Heyting algebras, $[\![\Gamma]\!]^{\mathcal{D}} \subseteq [\![A]\!]^{\mathcal{D}}$. Lem. 14 implies $\Gamma \in \lfloor\Gamma\rfloor = [\![\Gamma]\!]^{\mathcal{D}}$, so $\Gamma \in [\![A]\!]^{\mathcal{D}} = \lfloor A\rfloor$ and by Def. 13 there exists a proof-term π_0 such that $\Gamma \vdash \pi_0 : A$ and, in particular, $\pi_0 \in [\![A]\!]^{\mathcal{M}}_{\phi}$ for any ϕ. Therefore, π_0 is weakly normalizable. ◀

5.1 Towards normalization by evaluation

In order to obtain some algorithm of normalization by evaluation, we need to strengthen the result of Thm. 21. First we need to obtain some proof in normal form, instead of, in Thm. 21, a weakly normalizable proof:

$$\text{if } \Gamma \vdash A \text{ has a proof then } \Gamma \vdash A \text{ has a proof in normal form.} \quad (\mathrm{E}_1)$$

Then we need to relate this proof in normal form to the original one:

$$\text{if } \Gamma \vdash A \text{ has a proof } \pi \text{ then } \Gamma \vdash A \text{ has a proof in normal form } \pi_0 \text{ with } \pi \rhd^* \pi_0. \quad (\mathrm{E}_2)$$

Let us see how to obtain (E_1). We shall discuss about how to obtain (E_2) in the conclusion. In order to obtain (E_1) we refine Def. 13 as follows:

▶ **Definition 22** (strong outer value)**.** Let A be a formula. We let $\lfloor A\rfloor$ be the set of contexts Γ such that there exists some proof term π in ***normal form*** such that:

- $\Gamma \vdash \pi : A$
- for any environment ϕ, any assignment σ and any term-substitution θ, $\sigma\theta\pi \in [\![A]\!]^{\mathcal{M}}_{\phi}$.

We need, in that case, *WN*-reducibility candidates that satisfy stability by β-reduction (the (CR_2) property of usual reducibility candidates), as we shall see in the following. The algebra $\mathcal{B}$ contains *WN*-reducibility candidates (Def. 10) that do not enjoy stability by β-reduction. But we can build a sub-algebra $\mathcal{B}'$ that only contains sets of proof-terms that are stable by β-reduction.

▶ **Lemma 23** (Stability by reduction)**.** *Let $\mathcal{B}'$ be the smallest set of reducibility candidates closed by the operations of Def. 8. Let E be a member of $\mathcal{B}'$, π and π' be proof-terms. If $\pi \in E$ and $\pi \rhd^* \pi'$, then $\pi' \in E$.*

Proof. By induction on the construction of E. If $E = \tilde{\top}$ or $E = \tilde{\bot}$, this is immediate. Assume $E = F \mathrel{\tilde{\Rightarrow}} G$. If π is *WN* isolated, then so is π' (it is *WN* by confluence). Otherwise, $\pi \rhd^* \lambda\alpha.\pi_1$. By confluence, $\pi' \rhd^* \lambda\alpha.\pi_1'$ with $\pi_1 \rhd^* \pi_1'$. Let $\pi_2 \in F$. $(\pi_2/\alpha)\pi_1 \in G$ by definition of π_1 and $(\pi_2/\alpha)\pi_1 \rhd^* (\pi_2/\alpha)\pi_1'$. So by induction hypothesis, $(\pi_2/\alpha)\pi_1' \in G$, and $\pi' \in F \mathrel{\tilde{\Rightarrow}} G$. The other cases are handled similarly. ◀

This algebra $\mathcal{B}'$ is also full, ordered and complete so that we can also use super-consistency to build a model interpretation in it.
With this new algebra $\mathcal{B}'$, Lem. 14 and 17 hold without effort. We only need to reformulate the cases of Lem. 17.
Let us process a key case as an example: $\lfloor A\rfloor \leq \lfloor B\rfloor \mathrel{\check{\Rightarrow}} \lfloor C\rfloor$ only if $\lfloor A\rfloor \mathrel{\check{\wedge}} \lfloor B\rfloor \leq \lfloor C\rfloor$. Let $\Gamma \in \lfloor A\rfloor \cap \lfloor B\rfloor$. We build the proof:

$$\frac{\Gamma \vdash \pi : B \Rightarrow C \qquad \Gamma \vdash \pi_B : B}{\Gamma \vdash \pi\ \pi_B : C}$$

with $\pi \in [\![B \Rightarrow C]\!]^{\mathcal{M}}$ and $\pi_B \in [\![B]\!]^{\mathcal{M}}$, so that both are in normal form. However, $(\pi\ \pi_B)$ may not be in normal form. If this is not the case it means that $\pi = \lambda\alpha.\pi_1$. π is not isolated and by Def. 8 there exists π_1' such that $\pi \rhd^* \lambda\alpha.\pi_1'$ and $(\pi_B/\alpha)\pi_1' \in [\![C]\!]^{\mathcal{M}}$. Let π_C the normal form of $(\pi_B/\alpha)\pi_1'$.

Since $\sigma\theta(\pi\ \pi') \in [\![C]\!]^{\mathcal{M}}$ (exactly as in Lem. 17), and $\sigma\theta(\pi\ \pi') \rhd^* \sigma\theta\pi_C$, we have by Lem. 23 (so by stability by reduction) $\sigma\theta\pi_C \in [\![C]\!]^{\mathcal{M}}$ and π_C is the proof-term we wanted.

Now, Thm. 21 will produce proof-terms in *normal form*.

6 Conclusion and further work

In this paper, we have defined a notion of pre-models for weak normalization of theories expressed in intuitionistic natural deduction modulo. This notion of pre-model is defined as a model on a specific pre-Heyting algebra. To prove that the existence of such a pre-model implies cut elimination for the considered theory, we do not use the usual syntactic way, but we extract from such a pre-model, a (regular) model valued on some Heyting algebra, which implies cut elimination via the usual soundness/strong completeness paradigm. In the following, we discuss how to extend this result to obtain, in the same way, first weak normalization of the considered theory, and secondly a normalization by evaluation algorithm.

Thm. 21, does not state that all proofs of the sequent $\Gamma \vdash A$ are weakly normalizable, but the fact that if this sequent is provable (by a proof-term π), then there exists a weakly normalizable proof π_0 (or in normal form in Sec. 5.1) of that sequent. We have seen, in the previous section how to obtain that π_0 is not only weakly normalizable but also in normal form. But how to relate π_0 to π in order to obtain, first, a semantic proof that *WN*-reducibility candidates entail weak normalization, and second, a normalization by evaluation algorithm?

Indeed, in our construction, we have split the usual adequacy lemma [13] in two parts: the appeal to the proof-term and an inductive argument is handled by the *soundness* theorem, and the inductive cases, that are handled by Lem. 17. This way, we get rid of the original proof-term π and we have no way to get it back.

But if we formalize this algorithm, in Coq for instance, π will lift from the syntactic level towards the proof-assistant level. So it will be saved, and eventually be re-used to produce π_0 in Def. 13. However, there is still no way to show that π_0 is the normal form of π, unless we examine Coq's proof-term itself.

In order to enforce this, we would have to embed π at a lower level, as a *justification* of the fact that $\lfloor\Gamma\rfloor \subseteq \lfloor A\rfloor$ in the model, and to carry it in every proposition, especially Lem. 17.

It should be instructing to compare this approach to other normalization by evaluation approaches [1, 2, 3, 5, 6], in particular because they are all based on a Kripke-like structure, whereas in this paper we meet a pure Heyting algebra structure. Perhaps transforming Heyting algebras into Kripke structures [21], or a reverse operation like the one of [18] will help in this matter. Also, since Kripke worlds are formed of contexts, the presence of contexts in outer values should be a hint that such a transformation is possible.

References

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

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

3 Ulrich Berger and Helmut Schwichtenberg. An inverse to the evaluation functional for typed λ-calculus. In *Proceedings of the 6th Annual IEEE Symposium on Logic in Computer Science*, pages 202–211, 1991.

4 Richard Bonichon and Olivier Hermant. On constructive cut admissibility in deduction modulo. In Thorsten Altenkirch and Conor McBride, editors, *TYPES for proofs and programs*, volume 4502 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2007.

5 Catarina Coquand. From semantics to rules: A machine-assisted analysis. In *CSL*, LNCS 832, pages 91–105. Springer, 1993.

6 Thierry Coquand and Jean Gallier. A proof of strong normalization for the theory of constructions using a Kripke-like interpretation. In *Preliminary Proceedings 1st Intl. Workshop on Logical Frameworks*, 1990.

7 Denis Cousineau. *Models and Proof Normalization.* PhD thesis, École Polytechnique, 2009.

8 Denis Cousineau. On completeness of reducibility candidates as a semantic for strong normalization. In *Logical Methods in Computer Science*, 2012.

9 Gilles Dowek. Truth value algebras and normalization. In Thorsten Altenkirch and Conor McBride, editors, *TYPES for proofs and programs*, volume 4502 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2007.

10 Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Hol-lambda-sigma: an intentional first-order expression of higher-order logic. *Mathematical Structures in Computer Science*, 11:1–25, 2001.

11 Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31:33–72, 2003.

12 Gilles Dowek and Alexandre Miquel. Cut elimination for Zermelo's set theory. *manuscript*, 2007.

13 Gilles Dowek and Benjamin Werner. Proof normalization modulo. *The Journal of Symbolic Logic*, 68(4):1289–1316, December 2003.

14 Gilles Dowek and Benjamin Werner. Arithmetic as a theory modulo. In Jürgen Giesl, editor, *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 423–437. Springer, 2005.

15 Gerhard Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1934.

16 Jean-Yves Girard. Une extension de l'interprétation de Gödel à l'analyse et son application à l'élimination des coupures dans l'analyse et la théorie des types. In *Proceedings of the Second Scandinavian Logic Symposium (Univ. Oslo, Oslo, 1970)*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92. North-Holland, Amsterdam, 1971.

17 Olivier Hermant. Semantic cut elimination in the intuitionistic sequent calculus. In Pawel Urzyczyn, editor, *Typed Lambda-Calculi and Applications*, volume 3461 of *LNCS*, pages 221–233, Nara, Japan, 2005. Springer.

18 James Lipton and Mary De Marco. Completeness and cut elimination in Church's intuitionistic theory of types. *Journal of Logic and Computation*, 15:821–854, December 2005.

19 Colin Riba. Union of reducibility candidates for orthogonal constructor rewriting. In *CiE'08*, pages 498–510, 2008.

20 Helmut Schwichtenberg and Anne Sjerp Troelstra. *Basic Proof Theory.* Cambridge University Press, 1996.

21 Anne Sjerp Troelstra and Dirk van Dalen. *Constructivism in Mathematics, An Introduction.* North-Holland, 1988.

One-context Unification with STG-Compressed Terms is in NP*

Carles Creus[1], Adrià Gascón[1], and Guillem Godoy[1]

1 Universitat Politècnica de Catalunya, departament de Llenguatges i Sistemes Informàtics, Jordi Girona 1, Barcelona, Spain
ccreuslopez@gmail.com adriagascon@gmail.com ggodoy@lsi.upc.edu

Abstract

One-context unification is an extension of first-order term unification in which a variable of arity one standing for a context may occur in the input terms. This problem arises in areas like program analysis, term rewriting and XML processing and is known to be solvable in nondeterministic polynomial time. We prove that this problem can be solved in nondeterministic polynomial time also when the input is compressed using Singleton Tree Grammars (STG's). STG's are a grammar-based compression method for terms that generalizes the directed acyclic graph representation. They have been recently considered as an efficient in-memory representation for large terms, since several operations on terms can be performed efficiently on their STG representation without a prior decompression.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Term Unification; Compression; Grammars

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.149

Category Regular Research Paper

1 Introduction

Term unification is a basic operation in many areas of computer science, especially in those related to logic (see [1] for a survey). Unifying two terms s, t corresponds to find a substitution σ for the variables occurring in the equation $s \doteq t$ such that $\sigma(s) = \sigma(t)$ holds. The particular case when one of the terms has no occurrences of variables is called matching.

A simple case of term unification is called first-order unification. In this case variables occur at leaf positions and stand for terms. Both the first-order unification and matching problems have applications in the context of automated deduction, functional and logic programming, rewriting, and pattern matching. Moreover, several variants of first-order unification have been studied to tackle problems arising in those areas, see [1]. A remarkable extension is unification modulo theories. In this notion of unification, equality between terms is interpreted under equational theories such as associativity, commutativity, and distributivity, among others.

Another widely considered notion of unification allows variables of arity one standing for contexts, in addition to variables at leaf positions. This extension is called context unification,

* The authors were supported by Spanish Ministry of Education and Science by the FORMALISM project (TIN2007-66523). The first author was also supported by an FPI-UPC grant. The second author was also supported by an FPU grant from the Spanish Ministry of Education. The last author was also supported by the Spanish Ministry of Science and Innovation SweetLogics project (TIN2010-21062-C02-01).

which is a particular case of second-order unification. While second-order unification is known to be undecidable [8], decidability of context unification is still open. However, some interesting results have been found for more particular cases with application in computational linguistics [10, 4, 12]. One-context unification is the particular case when only one context variable, possibly with many occurrences, may appear in the input terms. This problem can be solved in nondeterministic polynomial time [6] but it is not known whether it is NP-hard. One of the motivations for this specific variant is its close relation to interprocedural program analysis [9]. The goal of this kind of analysis is to compute all simple invariants of imperative procedural programs. Other interesting applications of one-context unification and matching arise in searching/extracting information from tree data structures. For example, a simple matching equation of the form $F(s) \doteq t$, where F is the context variable, t is ground, and s may contain first-order variables but it does not contain occurrences of F, corresponds to search instances of s within t. In the context of XML processing, combinations of unificaton modulo commutativity and context unification are used for querying XML documents with a logic programming approach [2].

Other relevant variants of unification are related to term compression. Some applications dealing with trees, especially in the context of XML processing, require some kind of succint representation for terms. For this reason, in addition to the well-known Directed Acyclic Graph representation (DAG), several compressed in-memory representations have been developed [18, 15]. Grammar-based compression techniques were initially applied to words [17] and led to important results in string processing. Besides its direct applications in data compression, this kind of representations has also been useful as a technique for complexity analysis [11]. This compression mechanisms were extended from words to terms in [3] to introduce Singleton Tree Grammars (STG's), which allow to represent terms with exponential size and height in linear space. STG's are useful in the context of XML tree structure compression [15], tree automata [16], XPATH [14], and unification theory [12]. Moreover, polynomial time algorithms for first-order unification and matching for the case when the input is compressed using STG's have been developed and implemented [5, 7].

In this work we extend the result in [6] proving that one-context unification can be solved in nondeterministic polynomial time even in an STG-compressed setting.

2 Preliminaries

2.1 Terms, Contexts and Substitutions

A *ranked alphabet* $\mathcal{F}$ is a (finite) set of function symbols with arity. Symbols in $\mathcal{F}$ of arity 0, called *constants*, are denoted by a, b; nonconstant symbols are denoted by g, h. A *set of variables* $\mathcal{V}$ is a ranked alphabet containing symbols of arity 0, called *first-order variables* and denoted by x, y, and symbols of arity 1, called *context variables* and denoted by F. We assume that $\mathcal{F}$ and $\mathcal{V}$ are always disjoint. We use α to refer to an element of $\mathcal{F} \cup \mathcal{V}$. The set $\mathcal{T}(\mathcal{F}, \mathcal{V})$ of *terms* over $\mathcal{F}$ and $\mathcal{V}$ is the smallest set such that $\alpha(t_1, \ldots, t_m)$ is in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ whenever $\alpha \in \mathcal{F} \cup \mathcal{V}$, the arity of α is m, and $t_1, \ldots, t_m \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. By s, t, u we denote terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$. The set of variables occurring in a term t is denoted by $\mathsf{Vars}(t)$.

A *position* is a sequence of natural numbers. The symbol λ denotes the empty sequence, also called the *root position*, and $p.p'$ denotes the concatenation of the positions p and p'. The set of positions of a term t, denoted by $\mathsf{Pos}(t)$, is defined recursively as $\mathsf{Pos}(\alpha(t_1, \ldots, t_m)) = \{\lambda\} \cup \{i.p \mid i \in \{1, \ldots, m\} \wedge p \in \mathsf{Pos}(t_i)\}$. The *length* of a position is denoted by $|p|$. Note that $|\lambda| = 0$ and $|i.p| = 1 + |p|$ hold. A position p_1 is a *prefix* of a position p, denoted $p_1 \leq p$, if there is a position p_2 such that $p_1.p_2 = p$ holds. Also, p_1 is a *proper prefix* of p, denoted

$p_1 < p$, if $p_1 \leq p$ and $p_1 \neq p$ hold. Two positions p, p' are parallel if $p \not\leq p'$ and $p' \not\leq p$ hold.

The *size* of a term t is denoted by $|t|$ and defined as $|\mathsf{Pos}(t)|$. The *subterm* of a term t at a position $p \in \mathsf{Pos}(t)$, denoted $t|_p$, is defined recursively as $t|_\lambda = t$ and $\alpha(t_1,\ldots,t_m)|_{i.p} = t_i|_p$. The *replacement* of a term t at a position $p \in \mathsf{Pos}(t)$ by a term s, denoted $t[s]_p$, is defined recursively as $t[s]_\lambda = s$ and $\alpha(t_1,\ldots,t_m)[s]_{i.p} = \alpha(t_1,\ldots,t_i[s]_p,\ldots,t_m)$. $\alpha(t_1,\ldots,t_m)$.

A *context* is a term with exactly one occurrence of a special constant symbol $\bullet$, called *hole*. The set of contexts over a ranked alphabet $\mathcal{F}$ and a set of variables $\mathcal{V}$ is denoted $\mathcal{C}(\mathcal{F},\mathcal{V})$. We use c to denote a context in $\mathcal{C}(\mathcal{F},\mathcal{V})$. The operations on terms defined above are extended to contexts in the natural way. The *hole position* of a context c, denoted $\mathsf{hp}(c)$, is the position in $\mathsf{Pos}(c)$ labeled by $\bullet$. Given a context c and a term t, we define the term $c[t]$ as $c[t]_{\mathsf{hp}(c)}$. Similarly, given another context c', the *concatenation* cc' is the context $c[c']_{\mathsf{hp}(c)}$. Note that $\mathsf{hp}(cc') = \mathsf{hp}(c).\mathsf{hp}(c')$ holds. The *exponentiation* of a context c to a natural number e, denoted c^e, is the context recursively defined as $c^e = cc^{e-|\mathsf{hp}(c)|}$ if $e > |\mathsf{hp}(c)| > 0$, as $c^e = c[\bullet]_p$ if $|\mathsf{hp}(c)| \geq e$, where $p \leq \mathsf{hp}(c)$ and $|p| = e$, and as $c^e = \bullet$ otherwise. Note that $|\mathsf{hp}(c^e)| = e$ holds for any context c with $|\mathsf{hp}(c)| > 0$ and natural number e. We say that c is a *subcontext* of a term t if $c = t|_{p_1}[\bullet]_{p_2}$, for $p_1.p_2 \in \mathsf{Pos}(t)$.

A *substitution*, denoted by σ, θ, is a total function $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{C}(\mathcal{F},\mathcal{V})$ such that $\sigma(\alpha) \in \mathcal{T}(\mathcal{F},\mathcal{V})$ if α is a first-order variable and $\sigma(\alpha) \in \mathcal{C}(\mathcal{F},\mathcal{V})$ if α is a context variable. The domain of a substitution σ is usually considered to be the set of variables α such that $\sigma(\alpha) \neq \alpha$, i.e. $\mathsf{Dom}(\sigma) = \{\alpha \mid \sigma(\alpha) \neq \alpha\}$. For this reason, when defining a particular substitution σ we do not make explicit $\sigma(\alpha)$ for variables $\alpha \notin \mathsf{Dom}(\sigma)$. We also define the variables occurring in a substitution σ as $\mathsf{Vars}(\sigma) = \bigcup_{\alpha\in\mathsf{Dom}(\sigma)}\{\alpha\} \cup \mathsf{Vars}(\sigma(\alpha))$. Moreover, substitutions are extended to be mappings from terms to terms, i.e. $\sigma : \mathcal{T}(\mathcal{F},\mathcal{V}) \to \mathcal{T}(\mathcal{F},\mathcal{V})$, as follows: $\sigma(g(t_1,\ldots,t_n)) = g(\sigma(t_1),\ldots,\sigma(t_n))$ and $\sigma(F(t)) = \sigma(F)[\sigma(t)]$. In addition, substitutions are also extended, in a similar way, to be mappings from contexts to contexts, i.e. $\sigma : \mathcal{C}(\mathcal{F},\mathcal{V}) \to \mathcal{C}(\mathcal{F},\mathcal{V})$. The *composition* of σ and θ, denoted $\theta \circ \sigma$, is defined as $\{\alpha \mapsto \theta(\sigma(\alpha)) \mid \alpha \in \mathsf{Dom}(\sigma) \cup \mathsf{Dom}(\theta)\}$.

2.2 Singleton Tree Grammars

A *Singleton Tree Grammar (STG)* is a 4-tuple $G = \langle \mathcal{TN}, \mathcal{CN}, \Sigma, R\rangle$, where $\mathcal{TN}$ is a set of nonterminals of arity 0, called *term nonterminals* and denoted by T, $\mathcal{CN}$ is a set of nonterminals of arity 1, called *context nonterminals* and denoted by C, and Σ is a ranked alphabet, whose elements are called *terminals*. The sets $\mathcal{TN}$, $\mathcal{CN}$, and Σ are pairwise disjoint. We denote by N nonterminals in $\mathcal{TN} \cup \mathcal{CN}$. R is a finite set of rules of the form: $T \to \alpha(T_1,\ldots,T_m)$, $T \to C_1T_2$, $C \to \bullet$, $C \to C_1C_2$, $C \to \alpha(T_1,\ldots,T_{i-1},C_i,T_{i+1},\ldots,T_m)$, $T \to T_1$, $C \to C_1$, where α is a terminal of arity m. STG's are nonrecursive, i.e. the transitive closure of the derivational relation between nonterminals is terminating. Furthermore, for every nonterminal N there is exactly one rule having N as left-hand side. Last two conditions guarantee that every nonterminal of an STG generates exactly one term/context. Given a term t (context c) with occurrences of nonterminals, the derivation of t (resp. c) by G is an exhaustive iterated replacement of the nonterminals by the corresponding right-hand sides. The result is denoted as $w_{G,t}$ (resp. $w_{G,c}$). In the case of a nonterminal N we also say that N *generates* $w_{G,N}$. The *size* of an STG G, denoted $|G|$, is the number of nonterminals of G. The size of the representation of an STG G is bounded by $|G| \cdot (2+m)$, where m is the maximum arity of the terminals of G, but we use the other notion to ease the presentation.

Our definition of STG's is different from the one in [3], where nonterminals are allowed to generate contexts with several holes. Nevertheless, both notions were proven equivalent in [16] up to a polynomial transformation. In the present paper we use STG's for term representation. It allows to represent terms with exponential size and height in linear space. This is in contrast with DAG's, which only allow for exponential compression in size.

▶ **Example 2.1.** Let n be a natural number. We define the STG G_n to have the following set of rules: $\{T \to C_n T_a,\ T_a \to a,\ C_\bullet \to \bullet,\ C_0 \to g(C_\bullet),\ C_1 \to C_0 C_0,\ C_2 \to C_1 C_1,\ C_3 \to C_2 C_2,\ \ldots,\ C_n \to C_{n-1} C_{n-1}\}$. Note that $w_{G_n,T} = g^{2^n}(a)$, which would require exponential space both in an explicit and a DAG representation.

Several properties can be computed efficiently on STG's (see [5, 12]), e.g. computing $|w_{G,N}|$ for every nonterminal N of G or computing the symbol labeling a certain position of the generated term/context. A remarkable result used in the present paper is stated in the following lemma. The rest of known results on STG's needed in this work are presented in the following section.

▶ **Lemma 2.2** ([13, 3]). *Given an STG G and two term nonterminals T_1, T_2 of G, it is decidable in time $\mathcal{O}(|G|^3)$ whether $w_{G,T_1} = w_{G,T_2}$.*

2.3 Context Unification

Given two terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, the *context unification* problem consists of deciding whether s and t are unifiable, i.e. whether there exists a substitution $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V}) \cup \mathcal{C}(\mathcal{F}, \mathcal{V})$ such that $\sigma(s) = \sigma(t)$. The *one-context unification* problem is the particular case of context unification in which $\mathcal{V}$ contains exactly one context variable. *First-order unification* corresponds to the particular case where there are no context variables in $\mathcal{V}$. For a fixed ranked alphabet $\mathcal{F}$, we define an *instance* of the one-context unification problem as a triple $\langle \Delta, \mathcal{X}, F \rangle$, where $\mathcal{X}$ is a set of first-order variables, F is a context variable, and Δ is a set of equations, i.e. a set of unordered pairs, of the form $s \doteq t$, with $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \{F\})$. A solution of $\langle \Delta, \mathcal{X}, F \rangle$ is a substitution $\sigma : \mathcal{X} \cup \{F\} \to \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \{F\}) \cup \mathcal{C}(\mathcal{F}, \mathcal{X} \cup \{F\})$ such that $\forall (s \doteq t) \in \Delta \ : \ \sigma(s) = \sigma(t)$. The definition of an instance that considers a single equation and ours are equivalent, but considering a set of equations is more appropriate in our setting.

▶ **Example 2.3.** Consider the set of equations $\{F(a) \doteq g(a, x), F(b) \doteq g(x, b)\}$. It can be unified by the substitution $\{F \mapsto g(\bullet, b), x \mapsto b\}$. Now consider the equation $F(g(x, b)) \doteq g(a, F(y))$. It has infinitely many solutions such as $\{F \mapsto g(a, \bullet), x \mapsto a, y \mapsto b\}$ and $\{F \mapsto g(a, g(a, \bullet)), x \mapsto a, y \mapsto b\}$. Finally, consider the set of equations $\{F(a) \doteq g(x, y), F(b) \doteq g(x, g(g(y, y), g(a, b)))\}$, which has no solution.

By considering that the input terms are compressed using STG's, we obtain *one-context unification with STG's*. Fixed a ranked alphabet $\mathcal{F}$, an *instance* of this problem is a tuple $\langle \Delta, G, \mathcal{X}, F \rangle$, where $\mathcal{X}$ is a set of first-order variables, F is a context variable, $G = \langle \mathcal{TN}, \mathcal{CN}, \mathcal{F} \cup \mathcal{X} \cup \{F\}, R \rangle$ is an STG, and Δ is a set of equations of the form $\{S_1 \doteq T_1, \ldots, S_n \doteq T_n\}$, where the S_i's and the T_i's are term nonterminals of G. With this representation, the context variable and first-order variables are initially represented as unary and constant terminal symbols of the grammar, respectively. Given an instance $\langle \{S_1 \doteq T_1, \ldots, S_n \doteq T_n\}, G, \mathcal{X}, F \rangle$, its corresponding uncompressed one-context unification instance is $\langle \{w_{G,S_1} \doteq w_{G,T_1}, \ldots, w_{G,S_n} \doteq w_{G,T_n}\}, \mathcal{X}, F \rangle$.

3 Known Results

3.1 Operations on STG's

In this section we describe the operations needed to compute a solution to an STG-compressed one-context unification instance. The following definition of *extension of an STG* describes the result of those operations.

▶ **Definition 3.1.** Let $\mathcal{F}$ be a ranked alphabet. Let $G = \langle \mathcal{TN}, \mathcal{CN}, \Sigma, R\rangle$ be an STG such that $\mathcal{F} \subseteq \Sigma$. An $\mathcal{F}$*-extension* of G is an STG $G' = \langle \mathcal{TN}', \mathcal{CN}', \Sigma', R'\rangle$ such that $\mathcal{TN} \subseteq \mathcal{TN}'$, $\mathcal{CN} \subseteq \mathcal{CN}'$, $R \subseteq R'$, and $\mathcal{F} \subseteq \Sigma'$.

The goal of the previous definition is to capture operations on STG's that correspond to instantiation of variables. More concretely, let $G = \langle \mathcal{TN}, \mathcal{CN}, \mathcal{F} \cup \mathcal{V}, R\rangle$ be an STG and let $G' = \langle \mathcal{TN}', \mathcal{CN}', \mathcal{F} \cup \mathcal{V}', R'\rangle$ be an $\mathcal{F}$-extension of G. The terms generated by term nonterminals of G and G' are assumed to belong to the sets $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\mathcal{T}(\mathcal{F}, \mathcal{V}')$, respectively, and analogously for contexts. Hence, by defining a substitution $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V}') \cup \mathcal{C}(\mathcal{F}, \mathcal{V}')$ as $\sigma(\alpha) = w_{G',\alpha}$ it holds that, for each nonterminal N of G, $\sigma(w_{G,N}) = w_{G',N}$.

In [5] it is proven that first-order unification can be solved in polynomial time when the input terms are compressed using STG's. This approach proceeds analogously to the classical unification process. The main idea is that, given a first-order equation $s \doteq t$, the algorithm iteratively finds a position p labeled by a variable x in one of the terms, say s, and replaces all its occurrences by the corresponding subterm $t|_p$. This process ends when either s and t become equal, and thus they are unifiable, or a contradiction is reached. Since each iteration of the algorithm instantiates a variable, the solution σ can be described as an ordered sequence of substitutions on first-order variables. The authors focus on showing that this behaviour can be efficiently adapted to the STG-compressed setting. Besides checking equality at each iteration, the only operation that the algorithm needs to perform is to apply substitutions of the form $\{x \mapsto t|_p\}$. This corresponds to computing an $\mathcal{F}$-extension of the grammar that adds n new nonterminals $T_1, \ldots, T_n$ such that T_n generates $t|_p$ using $T_1, \ldots, T_{n-1}$ and adds a new rule $x \to T_n$, i.e. converting the terminal symbol x into a nonterminal generating $t|_p$. The crucial result to prove that such an $\mathcal{F}$-extension can be computed in polynomial time is that all such n are linearly bounded by the size of the initial input grammar and not by the size of the current grammar at each step of the process. Note that this implies that the size of the final grammar, i.e. the grammar obtained after all the variables have been replaced, is polynomially bounded by the size of the initial grammar. This technical fact is summarized in the following lemma, which is proven in [5].

▶ **Lemma 3.2.** *Let $G = \langle \mathcal{TN}, \mathcal{CN}, \Sigma, R\rangle$ be an STG describing a one-context unification instance. Let T be a term nonterminal of G and let t_0 be $w_{G,T}$. Let $t_1, \ldots, t_n$ be terms, $x_1, \ldots, x_n$ be first-order variables, $p_1, \ldots, p_n$ be positions, and $\sigma_1, \ldots, \sigma_n$ be substitutions satisfying, for $i \in \{1, \ldots, n\}$, $p_i \in \mathsf{Pos}(t_{i-1})$, $x_i \in \mathsf{Vars}(t_{i-1}) \setminus \mathsf{Vars}(t_{i-1}|_{p_i})$, $\sigma_i = \{x_i \mapsto t_{i-1}|_{p_i}\}$, and $t_i = \sigma_i(t_{i-1})$.*

Then, there exists an $\mathcal{F}$-extension $G' = \langle \mathcal{TN}', \mathcal{CN}', \Sigma \setminus \{x_1, \ldots, x_n\}, R'\rangle$ of G such that $w_{G',T} = t_n$ and $|G'| \leq |G| + n(|G| + 1)$.

In this paper we deal with one-context unification. Similar to the first-order case, instantiation of the special context variable F is performed by adding a rule $F \to C$ to the grammar, i.e. turning the terminal unary symbol F into a context nonterminal. This C is a new context nonterminal defined through the concatenation of several subcontexts of the input terms. In [5] it is shown how to efficiently compute subcontexts from a given STG-compressed term. The proposed construction guarantees that, given a grammar G, a nonterminal T and a position $p_1.p_2 \in \mathsf{Pos}(w_{G,T})$, G can be extended with, at most, $|G|(2|G| + 3)$ new nonterminals such that one of them generates the context $w_{G,T}|_{p_1}[\bullet]_{p_2}$. Moreover, given the context nonterminals $C_1, \ldots, C_n$ of a grammar G, an extended grammar containing a context nonterminal that generates the concatenation $w_{G,C_1} \ldots w_{G,C_n}$ can be easily obtained by adding n new nonterminals to G. These facts are stated in the following lemma, whose proof is also given in [5].

▶ **Lemma 3.3.** *Let $G = \langle \mathcal{TN}, \mathcal{CN}, \Sigma, R\rangle$ be an STG describing a one-context unification instance. Let T be a term nonterminal of G and let t be $w_{G,T}$. Let $p_1, \ldots, p_n, \hat{p}_1, \ldots, \hat{p}_n$ be positions and $c_1, \ldots, c_n$ be contexts satisfying, for $i \in \{1, \ldots, n\}$, $p_i {\cdot} \hat{p}_i \in \mathsf{Pos}(t)$ and $c_i = t|_{p_i}[\bullet]_{\hat{p}_i}$.*

Then, there exists an $\mathcal{F}$-extension $G' = \langle \mathcal{TN}', \mathcal{CN}', \Sigma, R'\rangle$ of G with a context nonterminal C such that $w_{G',C} = c_1 \ldots c_n$ and $|G'| \leq |G| + n|G|(2|G| + 3) + n$.

In some cases, the context nonterminal C that instantiates F is defined using one last construction: context exponentiation. Computing the grammar that generates the result of this operation is straightforward, as shown in the following illustrative example.

▶ **Example 3.4.** Let G be an STG with the following set of rules: $\{T_g \to g(T_h), T_h \to h(T_a), T_a \to a\}$. Note that w_{G,T_g} is $g(h(a))$. The context exponentiation $(w_{G,T_g}|_\lambda[\bullet]_{1.1})^{11}$ is $g(h(g(h(g(h(g(h(g(h(g(\bullet)\ldots)$ and can be generated by the STG with the set of rules:

$$\begin{aligned}&\{C \to g(C_h), C_h \to h(C_\bullet), C_\bullet \to \bullet\}\\ \cup\ &\{C_{\mathsf{exp4}} \to CC, C_{\mathsf{exp8}} \to C_{\mathsf{exp4}}C_{\mathsf{exp4}}, C_{\mathsf{exp10}} \to C_{\mathsf{exp8}}C\}\\ \cup\ &\{C_{\mathsf{pref}} \to g(C_\bullet)\}\\ \cup\ &\{C_{\mathsf{exp11}} \to C_{\mathsf{exp10}}C_{\mathsf{pref}}\}\end{aligned}$$

Note that we use the nonterminals C and C_{pref} to generate two different subcontexts, $g(h(\bullet))$ and $g(\bullet)$, respectively. Moreover, the nonterminals C_{exp4}, C_{exp8}, C_{exp10}, and C_{exp11} are used for exponentiation and concatenation, with C_{exp11} generating the desired context.

As seen in the previous example, raising a context to a natural number e requires to (i) compute two different subcontexts c_1 and c_2, (ii) concatenate c_1 with itself several times, and (iii) concatenate the resulting context with c_2. The construction done in (i) adds, at most, $|G|(2|G| + 3)$ for each computed subcontext, (ii) can be performed efficiently because the number of new nonterminals to be added is logarithmic with respect to e, and (iii) only adds one extra nonterminal to the grammar. This fact is stated formally in the following lemma.

▶ **Lemma 3.5.** *Let $G = \langle \mathcal{TN}, \mathcal{CN}, \Sigma, R\rangle$ be an STG describing a one-context unification instance. Let T be a term nonterminal of G and let t be $w_{G,T}$. Let p_1, p_2 be positions such that $p_1.p_2 \in \mathsf{Pos}(t)$ and let $e \geq 0$ be a natural number.*

Then, there exists an $\mathcal{F}$-extension $G' = \langle \mathcal{TN}', \mathcal{CN}', \Sigma, R'\rangle$ of G with a context nonterminal C such that $w_{G',C} = (t|_{p_1}[\bullet]_{p_2})^e$ and $|G'| \leq |G| + 2|G|(2|G| + 3) + \lceil \log_2(e)\rceil + 1$.

3.2 Uncompressed One-Context Unification

Consider an instance $\langle \Delta, \mathcal{X}, F\rangle$ of the one-context unification problem. In [6] it is proven that this problem is in NP when the terms in Δ are represented explicitly. The proposed algorithm is presented as an inference system that modifies the set of equations until a contradiction is found or the empty set is derived, which implies unifiability. By representing Δ with DAG's, the size of the representation of the terms in the set of equations is guaranteed to stay polynomially bounded by the size of the input at each step of a derivation. For the sake of clarity and in order to make this paper self contained, we introduce in Figure 1 a simplified version of that inference system. In this simplified version we have erased the rules used to early detect nonunifiability and some restrictions that eased the proofs by guaranteeing a bound on the length of every derivation. Hence, the new version of the inference system is simpler and still sound. Moreover, since it is less restrictive than the original one, each derivation leading to a solution can still be performed, and thus it is complete.

The most basic rule of the inference system is the rule Decompose, which is just used to simplify the unification problem. Note that it does not introduce any new subterms in

Decompose: $$\frac{\Delta = \Delta' \uplus \{\alpha(t_1,\ldots,t_n) \doteq \alpha(u_1,\ldots,u_n)\}}{\Delta' \cup \{t_1 \doteq u_1,\ldots,t_n \doteq u_n\}}$$

Var-Elim: $$\frac{x \doteq t \in \Delta}{\{x \mapsto t\}(\Delta)}$$ where $x \notin \mathsf{Vars}(t)$

Var-Elim2: $$\frac{F(u) \doteq c[x] \in \Delta}{\{x \mapsto F(\theta(u))\}(\theta(\Delta))}$$ where $\theta = \{F \mapsto c[F(\bullet)]\}$ and c is guessed such that $F \notin \mathsf{Vars}(c)$, $x \notin \mathsf{Vars}(u)$, and $(F \notin \mathsf{Vars}(u) \vee x \notin \mathsf{Vars}(c))$

CVar-Elim: $$\frac{F(u) \doteq c[t] \in \Delta}{\{F \mapsto c\}(\Delta)}$$ where c is guessed such that $F \notin \mathsf{Vars}(c)$

CVar-Elim2: $$\frac{F(u) \doteq c[F(t)] \in \Delta}{\{F \mapsto c^e\}(\Delta)}$$ where c is guessed such that $F \notin \mathsf{Vars}(c)$ and e is guessed such that $0 \le e \le 3\sum_{s \doteq t \in \Delta}(|s|+|t|)$

Figure 1 Inference system for one-context unification.

the set of equations. The remaining rules modify the current set of equations by replacing variables by subterms and subcontexts constructed from the terms in the set of equations. In particular, rule Var-Elim replaces a first-order variable by a term, rule Var-Elim2 partially guesses the initial part of F and instantiates a first-order variable, and rules CVar-Elim and CVar-Elim2 replace F by a context. As a technical detail, note that the substitution applied due to the application of rule Var-Elim2 can be seen as an instantiation of F in terms of a freshly introduced context variable, which, for clarity, we denote also as F. Finally, the rule CVar-Elim2 instantiates the context variable by a context raised to a natural number whose value is linearly bounded by the size of the current set of equations. Note that it is a bound on the exponent of periodicity of minimal solutions, i.e. the maximum number of periodic repetitions of a context in a minimal solution.

▶ **Example 3.6.** Consider the one-context unification instance $\langle\Delta, \{x\}, F\rangle$, where Δ contains only the following equation: $F(g(a, F(b))) \doteq g(x, g(a, x))$.

This instance has no solution. Note that neither Decompose, Var-Elim, nor CVar-Elim2 can be applied. In the case of CVar-Elim, we need to choose a position in $\mathsf{Pos}(g(x, g(a, x)))$ in order to guess a context c. Hence, there exist five different options for c: $c = \bullet$, $c = g(\bullet, g(a, x))$, $c = g(x, \bullet)$, $c = g(x, g(\bullet, x))$, and $c = g(x, g(a, \bullet))$. Note that, in any case, the resulting first-order equation after applying the substitution $\{F \mapsto c\}$ to Δ does not lead to a solution. Finally, Var-Elim2 cannot be applied since the left-hand side of the equation is of the form $F(u)$ and F occurs in u, and the right-hand side of the equation has two occurrences of x. In order to understand the conditions of rule Var-Elim2, note that they allow either x to occur more than once in the right-hand side or F to occur in u. The reason to consider these situations separately is that both facts cannot hold at the same time since it would lead to an instantiation of F in terms of itself, and thus a contradiction.

By the form of the rules, the following statement bounding the length of the derivations holds trivially.

▶ **Lemma 3.7.** *Let $\langle\Delta, \mathcal{X}, F\rangle$ be a one-context unification instance. Any derivation from Δ using the inference system of Figure 1 contains at most $|\mathcal{X}|$ occurrences of* Var-Elim *and* Var-Elim2*, and at most one occurrence of either* CVar-Elim *or* CVar-Elim2*.*

4 Approach

In our setting, the terms in the initial set of equations Δ are represented by an STG. In order to prove that one-context unification is also in NP in the STG-compressed case, we adapt the inference system described in Section 3.2. Since subterms, subcontexts, context exponentiation, and variable instantiations are known to be efficiently computable with STG's, as seen in Section 3.1, one may be tempted to simply reproduce the sequence of variable instantiations done in the uncompressed case. However, the size of the grammar may grow after certain operations and, in order to prove that it does not explode, we need to use a different approach. In particular, the difficulties rely on the fact that we do not have the analogous of Lemma 3.2 for successive partial instantiations of the context variable, which require to compute a subcontext and thus increase the size of the grammar.

Our approach consists of modifying the sequences of rule applications that describe a solution in order to guarantee that they can be represented in polynomial space using an STG. To simplify reasonings, we introduce in Figure 2 a new inference system $\mathfrak{R}$ that generalizes the previous one in Figure 1. In $\mathfrak{R}$ we assume without loss of generality that the initial set of equations $\Delta = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ is encoded as a single term. This can be done by extending the alphabet with new symbols $\mathfrak{d}$ and $\mathfrak{e}$ of arity n and 2, respectively, and defining $\mathsf{term}(\Delta) = \mathfrak{d}(\mathfrak{e}(s_1, t_1), \ldots, \mathfrak{e}(s_n, t_n))$. With this notion, the question of whether there exists a substitution σ, the solution for Δ, such that $\sigma(s_1) = \sigma(t_1), \ldots, \sigma(s_n) = \sigma(t_n)$ corresponds to check whether there exists a substitution σ, the solution for $\mathsf{term}(\Delta)$, such that $\sigma(\mathsf{term}(\Delta))$ is of the form $\mathfrak{d}(\mathfrak{e}(u_1, u_1), \ldots, \mathfrak{e}(u_n, u_n))$. This change in notation is useful to refer to subterms of both sides of the equations in Δ indistinctly as subterms of $\mathsf{term}(\Delta)$.

$$\mathrm{R_x}: \quad \frac{t}{\{x \mapsto t|_p\}(t)} \quad \text{where } p \in \mathsf{Pos}(t) \text{ and } x \in \mathsf{Vars}(t) \setminus \mathsf{Vars}(t|_p)$$

$$\mathrm{R_{FF}}: \quad \frac{t}{\{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}(t)} \quad \text{where } p_1.p_2 \in \mathsf{Pos}(t) \text{ and } F \in \mathsf{Vars}(t) \setminus \mathsf{Vars}(t|_{p_1})$$

$$\mathrm{R_{FC}}: \quad \frac{t}{\{F \mapsto (t|_{p_1}[\bullet]_{p_2})^e\}(t)} \quad \begin{array}{l}\text{where } p_1.p_2 \in \mathsf{Pos}(t),\ F \in \mathsf{Vars}(t) \setminus \mathsf{Vars}(t|_{p_1}[\bullet]_{p_2}),\\ \text{and } e \in \{0, \ldots, 3|t|\}\end{array}$$

Figure 2 The adapted inference system $\mathfrak{R}$.

It is easy to see that an application of Var-Elim, Var-Elim2, CVar-Elim, or CVar-Elim2 corresponds to the application of at most two of the rules of $\mathfrak{R}$. In particular, an application of Var-Elim is emulated by an application of $\mathrm{R_x}$. In the rest of this paper, we use $\mathrm{R_{xF}}$ and $\mathrm{R_{x\neg F}}$ to refer to applications of $\mathrm{R_x}$ in which the involved subterm $t|_p$ has an occurrence of F or not, respectively. An application of Var-Elim2 corresponds to an application of $\mathrm{R_{FF}}$ followed by an application of $\mathrm{R_{xF}}$. Finally, applications of CVar-Elim and CVar-Elim2 are emulated by $\mathrm{R_{FC}}$. The remaining original rule, Decompose, was only used to simplify the problem in order to apply other rules. Its behaviour is implicitly emulated in $\mathfrak{R}$ by defining its rules by means of subterms and subcontexts of t.

With $\rightarrow_{\mathrm{R_{xF}},x,p}$ and $\rightarrow_{\mathrm{R_{x\neg F}},x,p}$ we denote an application of rules $\mathrm{R_{xF}}$ and $\mathrm{R_{x\neg F}}$, respectively, making explicit the position p and the first-order variable x involved in the rule application. Analogously, with $\rightarrow_{\mathrm{R_{FF}},p_1,p_2}$ and $\rightarrow_{\mathrm{R_{FC}},p_1,p_2}$ we denote an application of $\mathrm{R_{FF}}$ and $\mathrm{R_{FC}}$, making explicit the positions p_1 and p_2 involved in the rule application. Sometimes, we do not make explicit x, p, p_1, and p_2 when they are clear from the context or not relevant. By $\rightarrow_{\mathfrak{R}}$ we denote the derivational relation using $\mathfrak{R}$ and, as usual, $\rightarrow_{\mathfrak{R}}^{+}$ denotes its transitive closure and $\rightarrow_{\mathfrak{R}}^{*}$ denotes its reflexive-transitive closure. Additionally, by $\rightarrow_r$ and $\rightarrow_r^*$, we denote the derivational relation using the rule r of $\mathfrak{R}$ and its reflexive-transitive closure, respectively.

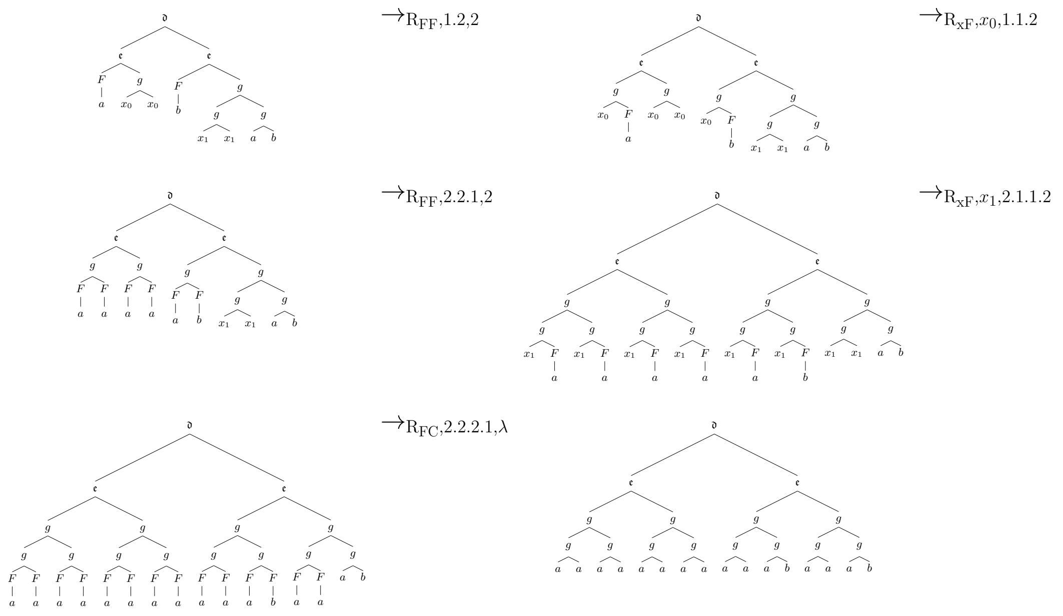
The following example illustrates the fact that $\mathfrak{R}$ can emulate derivations done with the inference system in Figure 1.

▶ **Example 4.1.** Consider the one-context unification instance $\langle\{F(a) \doteq g(x_0, x_0),\ F(b) \doteq g(g(x_1, x_1), g(a, b))\}, \{x_0, x_1\}, F\rangle$. In this example we use $\rightarrow_r^\sigma$ to denote a derivation step using the rule r and applying the substitution σ and $\rightarrow^*_{\mathsf{Decompose}}$ to denote some derivation steps using the rule Decompose. A possible successful derivation using the rules of Figure 1 is the following:

$$\begin{array}{l}
\{F(a) \doteq g(x_0, x_0),\ F(b) \doteq g(g(x_1, x_1), g(a, b))\} \\
\rightarrow^{\{x_0 \mapsto F(\{F \mapsto g(x_0, F(\bullet))\}(a))\} \circ \{F \mapsto g(x_0, F(\bullet))\}}_{\mathsf{Var-Elim2}} \\
\{g(F(a), F(a)) \doteq g(F(a), F(a)),\ g(F(a), F(b)) \doteq g(g(x_1, x_1), g(a, b))\} \\
\rightarrow^*_{\mathsf{Decompose}} \\
\{F(a) \doteq g(x_1, x_1),\ F(b) \doteq g(a, b)\} \\
\rightarrow^{\{x_1 \mapsto F(\{F \mapsto g(x_1, F(\bullet))\}(a))\} \circ \{F \mapsto g(x_1, F(\bullet))\}}_{\mathsf{Var-Elim2}} \\
\{g(F(a), F(a)) \doteq g(F(a), F(a)),\ g(F(a), F(b)) \doteq g(a, b)\} \\
\rightarrow^*_{\mathsf{Decompose}} \ \{F(a) \doteq a,\ F(b) \doteq b\} \ \rightarrow^{\{F \mapsto \bullet\}}_{\mathsf{CVar-Elim}} \ \{a \doteq a,\ b \doteq b\} \ \rightarrow^*_{\mathsf{Decompose}} \ \emptyset
\end{array}$$

Since we could derive $\emptyset$, the considered instance is indeed unifiable. The solution associated to the derivation is $\{F \mapsto g(g(a, a), g(a, \bullet)), x_0 \mapsto g(a, a), x_1 \mapsto a\}$.

We now show that the same solution can be derived also using $\mathfrak{R}$. The initial set of equations, expressed with the new notation, corresponds to the term $t = \mathsf{term}(\Delta) = \mathfrak{d}(\mathfrak{e}(F(a), g(x_0, x_0)), \mathfrak{e}(F(b), g(g(x_1, x_1), g(a, b))))$. Note that the subterm $t|_1$ encodes the equation $F(a) \doteq g(x_0, x_0)$ and $t|_2$ encodes $F(b) \doteq g(g(x_1, x_1), g(a, b))$. Our goal is to check whether there exists a substitution σ such that $\sigma(t)$ is of the form $\mathfrak{d}(\mathfrak{e}(t_1, t_1), \mathfrak{e}(t_2, t_2))$. The previous derivation corresponds to the following one using $\mathfrak{R}$:



$\rightarrow_{\mathrm{R_{FF}},1.2,2}$
$\rightarrow_{\mathrm{R_{xF}},x_0,1.1.2}$
$\rightarrow_{\mathrm{R_{FF}},2.2.1,2}$
$\rightarrow_{\mathrm{R_{xF}},x_1,2.1.1.2}$
$\rightarrow_{\mathrm{R_{FC}},2.2.2.1,\lambda}$

Note that, there are several equivalent options for the selection of the first position used in the last rule application. We chose 2.2.2.1 because this is the position that corresponds to the last steps of the previous derivation.

It is trivial that $\mathfrak{R}$ is sound. Moreover, since $\mathfrak{R}$ can emulate the original inference system, it follows that it is also complete. The following lemma states that soundness and completeness of $\mathfrak{R}$ also hold when the length of derivations is linearly bounded. This property follows from completeness of the inference system of Figure 1, soundness of $\mathfrak{R}$, and Lemma 3.7, taking into account that each rule of Figure 1 can be emulated with at most two rules of $\mathfrak{R}$.

▶ **Lemma 4.2.** *Let $\langle \Delta, \mathcal{X}, F\rangle$ be a one-context unification instance. Δ has a solution if and only if there exists a derivation $\mathsf{term}(\Delta) \rightarrow^*_{\mathfrak{R}} t$ of length at most $2|\mathcal{X}| + 1$ such that t is of the form $\mathfrak{d}(\mathfrak{e}(u_1, u_1), \ldots, \mathfrak{e}(u_{|\Delta|}, u_{|\Delta|}))$.*

In the case of one-context unification with STG's, $\mathsf{term}(\Delta)$ is represented by a single term nonterminal. More concretely, an instance of one-context unification with STG's $\langle\{S_1 \doteq T_1, \ldots, S_n \doteq T_n\}, \bar{G}, \mathcal{X}, F\rangle$ is represented as $\langle T, G, \mathcal{X}, F\rangle$, where G is an $\mathcal{F}$-extension of $\bar{G}$ and T is a term nonterminal of G such that $w_{G,T} = \mathfrak{d}(\mathfrak{e}(w_{\bar{G},S_1}, w_{\bar{G},T_1}), \ldots, \mathfrak{e}(w_{\bar{G},S_n}, w_{\bar{G},T_n})) = \mathsf{term}(\{w_{\bar{G},S_1} \doteq w_{\bar{G},T_1}, \ldots, w_{\bar{G},S_n} \doteq w_{\bar{G},T_n}\}) = \mathsf{term}(\Delta)$. With this new representation, the problem consists of deciding whether there exists a substitution σ such that $\sigma(w_{G,T})$ is of the form $\mathfrak{d}(\mathfrak{e}(u_1, u_1), \ldots, \mathfrak{e}(u_n, u_n))$. Recall that NP can be defined as the set of decisional problems whose positive instances can be verified in polynomial time. Hence, to prove that one-context unification with STG's is in NP we need to prove that there exists an $\mathcal{F}$-extension G' of G of polynomial size with respect to $|G|$ that represents σ. More concretely, in the rest of the paper we prove that, for each derivation of the form $\mathsf{term}(\Delta) \rightarrow^*_{\mathfrak{R}} t$ of length at most $2|\mathcal{X}| + 1$, there exists an $\mathcal{F}$-extension G' of G such that $w_{G',T} = t$ and whose size is polynomial with respect to $|G|$. This is enough for proving that one-context unification with STG's is in NP since the fact that t is of the form $\mathfrak{d}(\mathfrak{e}(u_1, u_1), \ldots, \mathfrak{e}(u_n, u_n))$ can be checked in polynomial time with respect to $|G'|$.

5 Commutation of Substitutions

As commented in the previous section, our approach consists of modifying the sequences of derivation steps with $\mathfrak{R}$ in such a way that allows to conclude that every unifiable instance $\langle \Delta, \mathcal{X}, F\rangle$ has a solution σ that can be represented in polynomial space using an STG. Our goal is to show that rule applications can always be commuted to obtain an equivalent derivation, i.e. a derivation describing the same substitution σ, that is of the following form:

$$\mathsf{term}(\Delta) \rightarrow^*_{\mathrm{R_{x\neg F}}} \rightarrow^*_{\mathrm{R_{FF}}} \rightarrow^*_{\mathrm{R_{xF}}} \rightarrow^{0,1}_{\mathrm{R_{FC}}} \rightarrow^*_{\mathrm{R_{x\neg F}}} \sigma(\mathsf{term}(\Delta))$$

where $\rightarrow^{0,1}_{\mathrm{R_{FC}}}$ denotes either 0 or 1 applications of the rule $\mathrm{R_{FC}}$. As a first ingredient in this argument, we define a particular notion of composition of substitutions which will be useful to reason in our setting. Next, we present some technical results to, finally, prove how to commute derivation steps until obtaining an equivalent derivation of the desired form.

5.1 Strong Composition

Consider two substitutions σ_1, σ_2. The goal of the following notion of composition of σ_1 and σ_2 is to capture how instantiations due to σ_1 are modified by the later application of σ_2.

▶ **Definition 5.1.** Let σ_1, σ_2 be substitutions. The *strong composition* of σ_1 and σ_2, denoted $\sigma_2 \diamond \sigma_1$, is defined as $\{\alpha \mapsto \sigma_2(\sigma_1(\alpha)) \mid \alpha \in \mathsf{Dom}(\sigma_1)\}$

The usual and the strong notions of composition are not equivalent in general. Recall that, given substitutions σ_1, σ_2, the usual notion of composition can be defined as $\sigma_2 \circ \sigma_1 =$

$\{\alpha \mapsto \sigma_2(\sigma_1(\alpha)) \mid \alpha \in \mathsf{Dom}(\sigma_1) \cup \mathsf{Dom}(\sigma_2)\}$. In order to stress the difference, consider $\theta_1 = \{y \mapsto b\}$ and $\theta_2 = \{x \mapsto a\}$ and note that $(\theta_2 \diamond \theta_1)(x) = x$, while $(\theta_2 \circ \theta_1)(x) = a$. Moreover, strong composition is not associative, i.e. $(\sigma_3 \diamond \sigma_2) \diamond \sigma_1 = \sigma_3 \diamond (\sigma_2 \diamond \sigma_1)$ does not hold in general: consider $\theta_0 = \{y \mapsto g(x)\}$ and note that $(\theta_2 \diamond \theta_1) \diamond \theta_0 = \{y \mapsto g(x)\}$, while $\theta_2 \diamond (\theta_1 \diamond \theta_0) = \{y \mapsto g(a)\}$. Another property that distinguishes both notions of composition is that, when using strong composition, $\mathsf{Dom}(\sigma_2 \diamond \sigma_1) \subseteq \mathsf{Dom}(\sigma_1)$ holds. This inclusion is strict only in anomalous cases, for example $\mathsf{Dom}(\{y \mapsto x\} \diamond \{x \mapsto y\}) = \emptyset$. In fact, the condition $\mathsf{Dom}(\sigma_2 \diamond \sigma_1) = \mathsf{Dom}(\sigma_1)$ is ensured when $\mathsf{Vars}(\sigma_2) \cap \mathsf{Dom}(\sigma_1) = \emptyset$, which is usually the case in our setting.

The following lemma is straightforward from the definition of strong composition.

▶ **Lemma 5.2.** *Let σ_1, σ_2 be substitutions and let α be a variable in $\mathsf{Dom}(\sigma_1)$. Then, $(\sigma_2 \circ \sigma_1)(\alpha) = (\sigma_2 \diamond \sigma_1)(\alpha)$.*

The following lemma states how two substitutions can be "commuted" using the strong composition.

▶ **Lemma 5.3.** *Let σ_1, σ_2 be substitutions such that $\mathsf{Vars}(\sigma_2) \cap \mathsf{Dom}(\sigma_1) = \emptyset$. Then, $\sigma_2 \circ \sigma_1 = (\sigma_2 \diamond \sigma_1) \circ \sigma_2$.*

Proof. Let $\mathcal{V}$ be a set of variables such that $\mathsf{Dom}(\sigma_1) \cup \mathsf{Dom}(\sigma_2) \subseteq \mathcal{V}$ holds. It suffices to prove that, for all $\alpha \in \mathcal{V}$, $(\sigma_2 \circ \sigma_1)(\alpha) = ((\sigma_2 \diamond \sigma_1) \circ \sigma_2)(\alpha)$ holds.

We distinguish cases depending on whether $\alpha \in \mathsf{Dom}(\sigma_1)$ and $\alpha \in \mathsf{Dom}(\sigma_2)$. If $\alpha \in \mathsf{Dom}(\sigma_1)$ then, by the assumption of the lemma, $\alpha \notin \mathsf{Dom}(\sigma_2)$ holds, and hence $((\sigma_2 \diamond \sigma_1) \circ \sigma_2)(\alpha) = (\sigma_2 \diamond \sigma_1)(\alpha) = (\sigma_2 \circ \sigma_1)(\alpha)$ holds by Lemma 5.2. If $\alpha \in \mathsf{Dom}(\sigma_2)$ then, by the assumption of the lemma, $\mathsf{Vars}(\sigma_2(\alpha)) \cap \mathsf{Dom}(\sigma_2 \diamond \sigma_1) = \emptyset$ and $\alpha \notin \mathsf{Dom}(\sigma_1)$ hold, and it follows $((\sigma_2 \diamond \sigma_1) \circ \sigma_2)(\alpha) = \sigma_2(\alpha) = (\sigma_2 \circ \sigma_1)(\alpha)$. Finally, the case where $\alpha \notin \mathsf{Dom}(\sigma_1) \cup \mathsf{Dom}(\sigma_2)$ trivially holds and the case where $\alpha \in \mathsf{Dom}(\sigma_1) \cap \mathsf{Dom}(\sigma_2)$ is not possible by the assumption. ◀

5.2 Subterm Preservation

Let t_1 and t_2 be terms such that t_2 is obtained from t_1 by applying a substitution. The following two lemmas state under which conditions a certain subterm of t_2 exists also as a subterm of t_1. These results will be crucial to argue about the commutation of derivation steps.

▶ **Lemma 5.4.** *Let $\mathcal{F}$ be a ranked alphabet and let $\mathcal{V}$ be a set with first-order variables and a context variable F. Let t_1, t_2 be terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $t_2 = \{F \mapsto c[F(\bullet)]\}(t_1)$, where c is a context in $\mathcal{C}(\mathcal{F}, \mathcal{V})$. Let p be a position in $\mathsf{Pos}(t_2)$ such that $F \notin \mathsf{Vars}(t_2|_p)$. Then, either there exists a position $\hat{p} \in \mathsf{Pos}(t_1)$ such that $t_1|_{\hat{p}} = t_2|_p$ or there exists a position $\hat{p} \in \mathsf{Pos}(c)$ such that $c|_{\hat{p}} = t_2|_p$.*

Proof. We prove the lemma by induction on $|p|$. If $p = \lambda$, note that F cannot occur in t_1 since, otherwise, $F \in \mathsf{Vars}(t_2|_p)$. Hence, in this case, $t_1 = t_2$ holds and we are done by defining $\hat{p}$ as p. For the induction step, assume that $|p| > 0$ and consider the following cases.

First, consider that t_1 is of the form $g(u_1, \ldots, u_n)$, where g is a function symbol in $\mathcal{F}$. Note that $n > 0$ holds since, otherwise, $t_2 = g$ and $p = \lambda$, contradicting the assumption. Let p be $i.p'$ more explicitly written, for $i \in \{1, \ldots, n\}$. Note that $t_2|_i = \{F \mapsto c[F(\bullet)]\}(u_i)$, $p' \in \mathsf{Pos}(t_2|_i)$, and $F \notin \mathsf{Vars}(t_2|_i|_{p'})$ hold. By induction hypothesis on $|p'|$, there exists a position $\hat{p}'$ such that either $u_i|_{\hat{p}'} = t_2|_i|_{p'}$ or $c|_{\hat{p}'} = t_2|_i|_{p'}$. The statement follows by defining $\hat{p}$ as $i.\hat{p}'$ in the former case and as $\hat{p}'$ in the latter case.

Second, consider that t_1 is of the form $F(u)$. We distinguish cases depending on whether p and $\mathsf{hp}(c)$ are parallel or not. Note that p cannot be a prefix of $\mathsf{hp}(c)$ since it would lead to a contradiction with the fact that $F \notin \mathsf{Vars}(t_2|_p)$. In the case where p and $\mathsf{hp}(c)$ are parallel, it follows that $c|_p = t_2|_p$, and we are done by defining $\hat{p}$ as p. Otherwise, let p be $\mathsf{hp}(c).1.p'$ more explicitly written. In this case, note that $t_2|_{\mathsf{hp}(c).1} = \{F \mapsto c[F(\bullet)]\}(u)$, $p' \in \mathsf{Pos}(t_2|_{\mathsf{hp}(c).1})$, and $F \notin \mathsf{Vars}(t_2|_{\mathsf{hp}(c).1}|_{p'})$ hold. Hence, by induction hypothesis on $|p'|$, there exists a position $\hat{p}'$ such that either $u|_{\hat{p}'} = t_2|_{\mathsf{hp}(c).1}|_{p'}$ or $c|_{\hat{p}'} = t_2|_{\mathsf{hp}(c).1}|_{p'}$. The statement follows by defining $\hat{p}$ as $1.\hat{p}'$ in the former case and as $\hat{p}'$ in the latter case.

Third, the case where t_1 is a first-order variable is not possible because, in such case, p must be λ, which contradicts the assumption. ◀

▶ **Lemma 5.5.** *Let $\mathcal{F}$ be a ranked alphabet and let $\mathcal{V}$ be a set with first-order variables and a context variable F. Let t_1, t_2 be terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that either $t_1 \rightarrow_{\mathrm{R_{xF}}} t_2$ or $t_1 \rightarrow_{\mathrm{R_{FF}}} t_2$. Let p be a position in $\mathsf{Pos}(t_2)$ such that $F \notin \mathsf{Vars}(t_2|_p)$. Then, there exists a position $\hat{p} \in \mathsf{Pos}(t_1)$ such that $t_1|_{\hat{p}} = t_2|_p$.*

Proof. We first consider the case where the applied rule is $\mathrm{R_{FF}}$. Let $t_1 \rightarrow_{\mathrm{R_{FF}},p_1,p_2} t_2$ be the derivation step of the statement more explicitly written. Hence, $t_2 = \{F \mapsto t_1|_{p_1}[F(\bullet)]_{p_2}\}(t_1)$ and, by Lemma 5.4, the statement holds.

Now assume that the applied rule is $\mathrm{R_{xF}}$ with variable x and position q. Let P be the subset of positions of $\mathsf{Pos}(t_1)$ labeled by x in t_1. Note that P is a set of pairwise parallel positions. Moreover, note that p cannot be a prefix of any of the positions in P since, otherwise, $F \in \mathsf{Vars}(t_2|_p)$, contradicting the assumptions of the lemma. In the case where p is parallel with every position in P, it follows that $p \in \mathsf{Pos}(t_1)$ and $t_1|_p = t_2|_p$. Otherwise, exactly one position $p' \in P$ is a proper prefix of p. Hence, p is of the form $p'.q'$ and it follows that $t_1|_{q.q'} = t_2|_p$. ◀

The following lemma states how the instantiation of a context variable and the computation of a subterm can be commuted.

▶ **Lemma 5.6.** *Let $\mathcal{F}$ be a ranked alphabet and let $\mathcal{V}$ be a set with first-order variables and a context variable F. Let t be a term in $\mathcal{T}(\mathcal{F}, \mathcal{V})$, let p be a position in $\mathsf{Pos}(t)$, and let $\sigma = \{F \mapsto c\}$ be a substitution, where c is a context in $\mathcal{C}(\mathcal{F}, \mathcal{V})$. Then, there exists a position $\hat{p} \in \mathsf{Pos}(\sigma(t))$ such that $\sigma(t|_p) = \sigma(t)|_{\hat{p}}$.*

Proof. We prove the lemma by induction on $|p|$. The base case, i.e. when $p = \lambda$, trivially holds by defining $\hat{p}$ as p. For the induction step, assume that $|p| > 0$. We distinguish cases depending on the form of t.

First, assume that t is of the form $g(u_1, \ldots, u_n)$, where g is a function symbol in $\mathcal{F}$. Note that $n > 0$ necessarily holds since, otherwise, $p = \lambda$, contradicting the assumption. Let p be $i.p'$ more explicitly written, for $i \in \{1, \ldots, n\}$. By induction hypothesis, there exists a position $\hat{p}' \in \mathsf{Pos}(\sigma(u_i))$ such that $\sigma(u_i|_{p'}) = \sigma(u_i)|_{\hat{p}'}$ holds. Hence, the statement holds by defining $\hat{p}$ as $i.\hat{p}'$.

Second, assume that t is of the form $F(u)$. Let p be $1.p'$ more explicitly written. By induction hypothesis, there exists a position $\hat{p}' \in \mathsf{Pos}(\sigma(u))$ such that $\sigma(u|_{p'}) = \sigma(u)|_{\hat{p}'}$ holds. Hence, the statement holds by defining $\hat{p}$ as $\mathsf{hp}(c).\hat{p}'$.

Third, the case where t is a first-order variable is not possible because, in such case, p must be λ, which contradicts the assumption. ◀

5.3 Reordering Derivations

In this section we show how derivations with $\mathfrak{R}$ can be modified in order to guarantee that rules are applied in a specific order. As basic ingredients, the following three technical lemmas show how pairs of derivation steps can be swapped.

▶ **Lemma 5.7.** *Let $\mathcal{F}$ be a ranked alphabet and let $\mathcal{V}$ be a set with first-order variables and a context variable F. Let t_1, t_2, t be terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $t_1 \to_{\mathrm{R_{xF}},x,p} t \to_{\mathrm{R_{FF}},p_1,p_2} t_2$. Then, there exists a position $\hat{p}_1 \in \mathsf{Pos}(t_1)$, a term $t' \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, and a position $\hat{p} \in \mathsf{Pos}(t')$ such that $t_1 \to_{\mathrm{R_{FF}},\hat{p}_1,p_2} t' \to_{\mathrm{R_{xF}},x,\hat{p}} t_2$ holds.*

Proof. By the conditions on the application of $\mathrm{R_{xF}}$, $x \in \mathsf{Vars}(t_1) \setminus \mathsf{Vars}(t_1|_p)$ and $F \in \mathsf{Vars}(t_1|_p)$ hold. In addition, since $\mathrm{R_{xF}}$ instantiates x, then $x \notin \mathsf{Vars}(t)$ holds. Moreover, by the conditions on the application of $\mathrm{R_{FF}}$, we know that $F \in \mathsf{Vars}(t) \setminus \mathsf{Vars}(t|_{p_1})$. Note that

$$\begin{aligned}
t_2 =& \{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}(\{x \mapsto t_1|_p\}(t_1)) && (1)\\
=& (\{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\} \diamond \{x \mapsto t_1|_p\})(\{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}(t_1)) && (2)\\
=& \{x \mapsto \{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}(t_1|_p)\}(\{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}(t_1)) && (3)\\
=& \{x \mapsto \{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1|_p)\}(\{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)) && (4)\\
=& \{x \mapsto \{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)|_{\hat{p}}\}(\{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)) && (5)
\end{aligned}$$

where (1) follows from definition of $\mathrm{R_{xF}}$ and $\mathrm{R_{FF}}$, (2) follows from Lemma 5.3, which can be applied because $\mathsf{Vars}(\{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}) \cap \mathsf{Dom}(\{x \mapsto t_1|_p\}) = \emptyset$ holds since $x \notin \mathsf{Vars}(t)$ and $x \neq F$, (3) follows from Definition 5.1, in (4) the implicit definition of $\hat{p}_1$ holding $t_1|_{\hat{p}_1} = t|_{p_1}$ follows from Lemma 5.5, which can be applied since $t_1 \to_{\mathrm{R_{xF}},x,p} t$ and $F \notin \mathsf{Vars}(t|_{p_1})$, and in (5) the implicit definition of $\hat{p}$ holding $\{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)|_{\hat{p}} = \{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1|_p)$ follows from Lemma 5.6.

Finally, let the t' of the lemma be defined as $\{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)$ and note that $t_2 = \{x \mapsto t'|_{\hat{p}}\}(t')$. It remains to prove that t' can be derived from t_1, and that t_2 can be derived from t', as stated in the lemma. First, note that $t_1 \to_{\mathrm{R_{FF}},\hat{p}_1,p_2} t'$ holds because F does not occur in $t_1|_{\hat{p}_1}$, since $t_1|_{\hat{p}_1} = t|_{p_1}$, and, moreover, $F \in \mathsf{Vars}(t_1)$ holds because $F \in \mathsf{Vars}(t)$ and $\mathsf{Vars}(t) \subsetneq \mathsf{Vars}(t_1)$. Now, note that $t' \to_{\mathrm{R_{xF}},x,\hat{p}} t_2$ holds because:

- $x \in \mathsf{Vars}(t')$, which follows from the facts that $x \in \mathsf{Vars}(t_1)$ and $\mathsf{Vars}(t') = \mathsf{Vars}(t_1)$,
- $x \notin \mathsf{Vars}(t'|_{\hat{p}})$, since $x \notin \mathsf{Vars}(t_1|_p)$, $x \notin \mathsf{Vars}(t)$, and $t'|_{\hat{p}} = \{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)|_{\hat{p}} = \{F \mapsto t|_{p_1}[F(\bullet)]_{p_2}\}(t_1|_p)$, and
- $F \in \mathsf{Vars}(t'|_{\hat{p}})$, since $F \in \mathsf{Vars}(t_1|_p)$ holds and thus $F \in \mathsf{Vars}(\{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1|_p)) = \mathsf{Vars}(\{F \mapsto t_1|_{\hat{p}_1}[F(\bullet)]_{p_2}\}(t_1)|_{\hat{p}}) = \mathsf{Vars}(t'|_{\hat{p}})$ also holds.

Hence, $t_1 \to_{\mathrm{R_{FF}},\hat{p}_1,p_2} t' \to_{\mathrm{R_{xF}},x,\hat{p}} t_2$ holds, which concludes the proof. ◀

The proofs of the following two lemmas are very similar and, due to lack of space, the second one is omitted.

▶ **Lemma 5.8.** *Let $\mathcal{F}$ be a ranked alphabet and let $\mathcal{V}$ be a set with first-order variables and a context variable F. Let t_1, t_2, t be terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $t_1 \to_{\mathrm{R_{FF}},p_1,p_2} t \to_{\mathrm{R_{x\neg F}},x,p} t_2$. Then, there exists a position $\hat{p} \in \mathsf{Pos}(t_1)$ and a term $t' \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $t_1 \to_{\mathrm{R_{x\neg F}},x,\hat{p}} t' \to_{\mathrm{R_{FF}},p_1,p_2} t_2$ holds.*

Proof. By the conditions on the application of rule $\mathrm{R_{FF}}$, $F \in \mathsf{Vars}(t_1) \setminus \mathsf{Vars}(t_1|_{p_1})$ holds. Moreover, by the conditions on the application of rule $\mathrm{R_{x\neg F}}$, we know that

$x \in \mathsf{Vars}(t) \setminus \mathsf{Vars}(t|_p)$ and $F \notin \mathsf{Vars}(t|_p)$. Note that

$$
\begin{aligned}
t_2 =& \{x \mapsto t|_p\}(\{F \mapsto t_1|_{p_1}[F(\bullet)]_{p_2}\}(t_1)) && (1)\\
=& (\{x \mapsto t|_p\} \diamond \{F \mapsto t_1|_{p_1}[F(\bullet)]_{p_2}\})(\{x \mapsto t|_p\}(t_1)) && (2)\\
=& \{F \mapsto \{x \mapsto t|_p\}(t_1|_{p_1}[F(\bullet)]_{p_2})\}(\{x \mapsto t|_p\}(t_1)) && (3)\\
=& \{F \mapsto \{x \mapsto t_1|_{\hat{p}}\}(t_1|_{p_1}[F(\bullet)]_{p_2})\}(\{x \mapsto t_1|_{\hat{p}}\}(t_1)) && (4)\\
=& \{F \mapsto \{x \mapsto t_1|_{\hat{p}}\}(t_1)|_{p_1}[F(\bullet)]_{p_2}\}(\{x \mapsto t_1|_{\hat{p}}\}(t_1)) && (5)
\end{aligned}
$$

where (1) follows from definition of $\mathrm{R_{FF}}$ and $\mathrm{R_{x\neg F}}$, (2) follows from Lemma 5.3, which can be applied because $\mathsf{Vars}(\{x \mapsto t|_p\}) \cap \mathsf{Dom}(\{F \mapsto t_1|_{p_1}[F(\bullet)]_{p_2}\}) = \emptyset$ holds since $F \notin \mathsf{Vars}(t|_p)$ and $F \neq x$, (3) follows from Definition 5.1, in (4) the implicit definition of $\hat{p}$ holding $t_1|_{\hat{p}} = t|_p$ follows from Lemma 5.5, which can be applied since $t_1 \to_{\mathrm{R_{FF}},p_1,p_2} t$ and $F \notin \mathsf{Vars}(t|_p)$, and (5) holds because replacements of first-order variables and computation of subterms can be commuted in this way.

Finally, let the t' of the lemma be defined as $\{x \mapsto t_1|_{\hat{p}}\}(t_1)$ and note that $t_2 = \{F \mapsto t'|_{p_1}[F(\bullet)]_{p_2}\}(t')$. It remains to prove that t' can be derived from t_1, and that t_2 can be derived from t', as stated in the lemma. First, note that $t_1 \to_{\mathrm{R_{x\neg F}},x,\hat{p}} t'$ holds because neither F nor x occur in $t_1|_{\hat{p}}$ since $t_1|_{\hat{p}} = t|_p$ and, moreover, $x \in \mathsf{Vars}(t_1)$ holds because $x \in \mathsf{Vars}(t)$ and $\mathsf{Vars}(t) = \mathsf{Vars}(t_1)$. Now, note that $t' \to_{\mathrm{R_{FF}},p_1,p_2} t_2$ holds because:

- $F \in \mathsf{Vars}(t')$, since $F \in \mathsf{Vars}(t_1) \setminus \{x\} = \mathsf{Vars}(\{x \mapsto t_1|_{\hat{p}}\}(t_1)) = \mathsf{Vars}(t')$, and
- $F \notin \mathsf{Vars}(t'|_{p_1})$ since $F \notin \mathsf{Vars}(t_1|_{p_1}) \cup \mathsf{Vars}(\{x \mapsto t_1|_{\hat{p}}\})$ and thus $F \notin \mathsf{Vars}(\{x \mapsto t_1|_{\hat{p}}\}(t_1|_{p_1})) = \mathsf{Vars}(\{x \mapsto t_1|_{\hat{p}}\}(t_1)|_{p_1}) = \mathsf{Vars}(t'|_{p_1})$.

Hence, $t_1 \to_{\mathrm{R_{x\neg F}},x,\hat{p}} t' \to_{\mathrm{R_{FF}},p_1,p_2} t_2$ holds, which concludes the proof. ◀

▶ **Lemma 5.9.** *Let $\mathcal{F}$ be a ranked alphabet and let $\mathcal{V}$ be a set with first-order variables and a context variable F. Let t_1, t_2, t be terms in $\mathcal{T}(\mathcal{F},\mathcal{V})$ such that $t_1 \to_{\mathrm{R_{xF}},x_1,p_1} t \to_{\mathrm{R_{x\neg F}},x_2,p_2} t_2$. Then, there exists a position $\hat{p}_2 \in \mathsf{Pos}(t_1)$ and a term $t' \in \mathcal{T}(\mathcal{F},\mathcal{V})$ such that $t_1 \to_{\mathrm{R_{x\neg F}},x_2,\hat{p}_2} t' \to_{\mathrm{R_{xF}},x_1,p_1} t_2$ holds.*

The following result follows from the three previous lemmas, summarizing the goal of this section.

▶ **Lemma 5.10.** *Let $\langle \Delta, \mathcal{X}, F\rangle$ be a one-context unification instance. Let t be a term such that $\mathsf{term}(\Delta) \to^*_{\mathfrak{R}} t$ in n derivation steps. Then, there exists a derivation of length n of the form $\mathsf{term}(\Delta) \to^*_{\mathrm{R_{x\neg F}}} \to^*_{\mathrm{R_{FF}}} \to^*_{\mathrm{R_{xF}}} \to^{0,1}_{\mathrm{R_{FC}}} \to^*_{\mathrm{R_{x\neg F}}} t$.*

6 Complexity Analysis

We now have all the ingredients needed to prove that the one-context unification where the input terms are compressed using STG's is in NP. To prove this fact, we show, for each unifiable instance, that there exists a witness of polynomial size verifiable in polynomial time. In our setting, this witness is an STG generating the unified term and the verification consists of checking whether such a term is of a certain form.

▶ **Theorem 6.1.** *One-context unification with STG's is in NP.*

Proof. Let $\langle \Delta, \mathcal{X}, F\rangle$ be a one-context unification instance represented by an STG G and a nonterminal T of G such that $w_{G,T} = \mathsf{term}(\Delta)$. By Lemma 4.2, Δ has a solution if and only if there exists a derivation $\mathsf{term}(\Delta) \to^*_{\mathfrak{R}} t$ of length at most $2|\mathcal{X}|+1$ such that t is of the form $\mathfrak{d}(\mathfrak{e}(u_1,u_1),\ldots,\mathfrak{e}(u_{|\Delta|},u_{|\Delta|}))$. Moreover, by Lemma 5.10, we assume without loss of generality that this derivation is of the form $\mathsf{term}(\Delta) \to^*_{\mathrm{R_{x\neg F}}} t_1 \to^*_{\mathrm{R_{FF}}} t_2 \to^*_{\mathrm{R_{xF}}} t_3 \to^{0,1}_{\mathrm{R_{FC}}} t_4 \to^*_{\mathrm{R_{x\neg F}}} t$, for some terms t_1, t_2, t_3, t_4.

We first prove that there exists an $\mathcal{F}$-extension G' of G such that $w_{G',T} = t$ and whose size is polynomially bounded by $|G|$. By Lemma 3.2, there exists an $\mathcal{F}$-extension G_1 of G such that $w_{G_1,T} = t_1$ and whose size is polynomially bounded by $|G|$. Now we claim that there exists an $\mathcal{F}$-extension G_2 of G_1 such that $w_{G_2,T} = t_2$ and whose size is polynomially bounded by $|G_1|$. By the conditions on the application of $\mathrm{R_{FF}}$ and by Lemma 5.5, the subcontexts computed in each step of the subderivation $t_1 \to^*_{\mathrm{R_{FF}}} t_2$ can be obtained from t_1. Hence, the sequence of applications of rules $\mathrm{R_{FF}}$ can be seen as a single application of a substitution of the form $\{F \mapsto c_1 \ldots c_n F(\bullet)\}$, where each c_i is a subcontext of t_1. By Lemma 3.3, there exists an $\mathcal{F}$-extension G'_1 of G_1 such that $w_{G'_1,T} = t_1$ with a context nonterminal C such that $w_{G'_1,C} = c_1 \ldots c_n$ and whose size is polynomially bounded by $|G_1|$ and $|\mathcal{X}|$. The STG G_2 mentioned above is defined as the $\mathcal{F}$-extension of G'_1 obtained by transforming the context variable into a context nonterminal generating $c_1 \ldots c_n F(\bullet)$, where F is a freshly introduced terminal symbol standing for the context variable. Again by Lemma 3.2, there exists an $\mathcal{F}$-extension G_3 of G_2 such that $w_{G_3,T} = t_3$ and whose size is polynomially bounded by $|G_2|$. At this point, note that the exponent e involved in the application of $\mathrm{R_{FC}}$ can be at most exponential with respect to $|G_3|$ since it is linear with $|t_3|$. Hence, by Lemma 3.5, there exists an $\mathcal{F}$-extension G_4 of G_3 such that $w_{G_4,T} = t_4$ and whose size is polynomially bounded by $|G_3|$. Hence, by definition of G_1, G_2, G_3, G_4, it follows that $|G_4|$ is polynomially bounded by $|G|$. Finally, the existence of the grammar G' mentioned above follows from Lemma 3.2 and the fact that $|G_4|$ is polynomially bounded by $|G|$. Note that, as commented in Section 2.2, the size of the representation of G' is bounded by $|G'| \cdot (2 + m)$, where m is the maximum arity of the terminals of G'.

To conclude the proof, note that the property whether the term generated by a certain nonterminal, in our case by the nonterminal T of G', is of the form $\mathfrak{d}(\mathfrak{e}(u_1, u_1), \ldots, \mathfrak{e}(u_{|\Delta|}, u_{|\Delta|}))$ can be checked in polynomial time with respect to the size of the given grammar. To see this, first note that checking whether the symbol labeling $w_{G',T}$ at position p, with $|p| = 0$ or $|p| = 1$, is $\mathfrak{d}$ or $\mathfrak{e}$, respectively, can be computed in linear time. Finally, nonterminals generating $w_{G',T}|_{i.1}$ and $w_{G',T}|_{i.2}$, for $i \in \{1, \ldots, |\Delta|\}$, can be computed efficiently, and thus, by Lemma 2.2, checking whether $w_{G',T}|_{i.1} = w_{G',T}|_{i.2}$ can be solved in polynomial time. ◀

7 Conclusions and Further Work

We have proved that the one-context unification problem belongs to NP even when the input is compressed using STG's. A natural next step is to study whether there exists a polynomial time algorithm for this problem. However, this is also open when using an uncompressed term representation. Hence, it seems reasonable to first consider this problem before tackling the compressed case.

Another option is to study the complexity of particular cases of one-context unification with STG's. Concretely, the following one is particularly interesting due to its applications in term rewriting and querying XML-databases: solving an equation $F(s) \doteq t$ where t is ground and s does not contain the context variable F but may contain first-order variables. As shown in [19], this problem can be solved in polynomial time when the representation of s is noncompressing. However, it is still open whether the general case is NP-hard or it can be solved in polynomial time.

References

1 F. Baader and W. Snyder. Unification theory. In *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.

2 F. Bry and S. Schaffert. Towards a declarative query and transformation language for xml and semistructured data: Simulation unification. In *ICLP*, pages 255–270, 2002.

3 G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Information Systems*, 33(4-5):456–474, 2008.

4 K. Erk and J. Niehren. Dominance constraints in stratified context unification. *Information Processing Letters*, 101(4):141–147, 2007.

5 A. Gascón, G. Godoy, and M. Schmidt-Schauß. Unification and matching on compressed terms. *ACM Transactions on Computational Logic*, 12(4):26, 2011.

6 A. Gascón, G. Godoy, M. Schmidt-Schauß, and A. Tiwari. Context unification with one context variable. *Journal of Symbolic Computation*, 45(2):173–193, 2010.

7 A. Gascón, S. Maneth, and L. Ramos. First-order unification on compressed terms. In *RTA*, pages 51–60, 2011.

8 W. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

9 S. Gulwani and A. Tiwari. Computing procedure summaries for interprocedural analysis. In *ESOP*, pages 253–267, 2007.

10 J. Levy, J. Niehren, and M. Villaret. Well-nested context unification. In *CADE*, pages 149–163, 2005.

11 J. Levy, M. Schmidt-Schauß, and M. Villaret. The complexity of monadic second-order unification. *SIAM Journal on Computing*, 38(3):1113–1140, 2008.

12 J. Levy, M. Schmidt-Schauß, and M. Villaret. On the complexity of bounded second-order unification and stratified context unification. *Logic Journal of the IGPL*, 19(6):763–789, 2011.

13 Y. Lifshits. Processing compressed texts: A tractability border. In *CPM*, pages 228–240, 2007.

14 M. Lohrey and S. Maneth. The complexity of tree automata and XPath on grammar-compressed trees. *Theoretical Computer Science*, 363(2):196–210, 2006.

15 M. Lohrey, S. Maneth, and R. Mennicke. Tree structure compression with RePair. In *DCC*, pages 353–362, 2011.

16 M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction in grammar-compressed trees. In *FOSSACS*, pages 212–226, 2009.

17 W. Plandowski. Testing equivalence of morphisms on context-free languages. In *ESA*, pages 460–470, 1994.

18 K. Sadakane and G. Navarro. Fully-functional succinct trees. In *SODA*, pages 134–149, 2010.

19 M. Schmidt-Schauß. Pattern matching of compressed terms and contexts and polynomial rewriting. Frank report 43, Institut für Informatik. Goethe-Universität Frankfurt am Main, February 2011.

Deciding Confluence of Ground Term Rewrite Systems in Cubic Time*

Bertram Felgenhauer[1]

1 **Institute of Computer Science, University of Innsbruck, Austria**
`bertram.felgenhauer@uibk.ac.at`

Abstract

It is well known that the confluence property of ground term rewrite systems (ground TRSs) is decidable in polynomial time. For an efficient implementation, the degree of this polynomial is of great interest. The best complexity bound in the literature is given by Comon, Godoy and Nieuwenhuis (2001), who describe an $O(n^5)$ algorithm, where n is the size of the ground TRS. In this paper we improve this bound to $O(n^3)$. The algorithm has been implemented in the confluence tool CSI.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems – Decision problems

Keywords and phrases confluence, term rewriting, decidability, ground terms

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.165

Category Regular Research Paper

1 Introduction

It is well known that confluence of ground TRSs can be decided in polynomial time. In this paper, we are interested in the degree of the associated polynomial.

To derive a polynomial time decision procedure for confluence of ground TRSs, Comon et al. [3] use an approach based on a transformation by Plaisted [9] that flattens the TRS. Then they test *deep joinability* of sides of rules. The authors sketch an implementation with complexity $O(n^5)$, where n is the size of the given TRS. Tiwari [10] and Godoy et al. [6] base their approach on a rewrite closure that constructs tree transducers—the given TRS $\mathcal{R}$ is converted into two TRSs $\mathcal{F}$ and $\mathcal{B}$ such that $\mathcal{F}$ and $\mathcal{B}^{-1}$ are left-flat, right-constant, $\mathcal{F}$ is terminating, and $\to_{\mathcal{R}}^* = \to_{\mathcal{F}}^* \cdot \to_{\mathcal{B}}^*$. They then consider *top-stabilizable* terms to derive conditions for confluence. Tiwari obtains a bound of $O(n^9)$ (but a more careful implementation would end up with $O(n^6)$), while Godoy et al. obtain a bound of $O(n^6)$. The algorithm of [3] is limited to ground TRSs, but [10] extends the algorithm to certain shallow, linear systems, and [5] treats shallow, linear systems in full generality.[1] In these extensions, however, the exponent depends on the maximum arity of the function symbols of the given TRS. In our work we combine ideas from [3, 10, 6] in order to improve the complexity bound to $O(n^3)$. The key ingredients are a Plaisted-style rewrite closure, which results in TRSs $\mathcal{F}$ and $\mathcal{B}$ of only quadratic size, and top-stabilizability, which is cheaper to test than deep joinability.

* This research is supported by FWF (Austrian Science Fund) project P22467.
[1] The same claim can be found in [6]. However, rule splitting, a key step in the proof of their Lemma 3.1, only works if left-hand side and right-hand side variables are disjoint for every rule.

The remainder of this paper is structured as follows: After some preliminaries in Section 2 we describe the confluence check in Section 3. Some experimental results are presented in Section 4. Finally we conclude in Section 5.

2 Preliminaries

A signature is a set of function symbols $\mathcal{F} = \mathcal{F}^{(0)} \cup \mathcal{F}^{(1)} \cup \ldots$, where $\mathcal{F}^{(i)}$ is the set of function symbols of arity i, and the sets $(\mathcal{F}^{(i)})_{i \in \mathbb{N}}$ are pairwise disjoint. The ground terms $\mathcal{T}(\mathcal{F})$ over $\mathcal{F}$ are constructed inductively in the usual way: If $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ and $f \in \mathcal{F}^{(n)}$, then $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F})$. A position p of a term is a sequence of natural numbers addressing a subterm $t|_p$. Replacement of subterms $t[u]_p$, and the size of terms $|t|$ have their standard definitions [2]. A term t together with a position p defines a context $C[\cdot] = t[\cdot]_p$. Contexts can be instantiated, $C[s] = t[s]_p$. The elements of $\mathcal{F}^{(0)} \subseteq \mathcal{F}$ are called constants. A term is flat if it is either a constant or a function symbol applied to constants.

A set $\mathcal{R} \subseteq \mathcal{T}(\mathcal{F})^2$ of rules is a ground term rewrite system (TRS). If $(\ell, r) \in \mathcal{R}$, we also write $\ell \to r \in \mathcal{R}$ and sometimes $\ell \approx r \in \mathcal{R}$. By $\mathcal{R}^-$, $\mathcal{R}^\pm$, $|\mathcal{R}|$, $\|\mathcal{R}\|$ we denote $\mathcal{R}^{-1}$ (where we view $\mathcal{R}$ as a relation on ground terms), $\mathcal{R} \cup \mathcal{R}^{-1}$, the number of rules in $\mathcal{R}$, and the total size of the rules, $\sum_{\ell \to r \in \mathcal{R}} (|\ell| + |r|)$, respectively. Any ground TRS $\mathcal{R}$ induces a rewrite relation $\to_\mathcal{R}$ on ground terms: $s \to_\mathcal{R} t$ if there is a context $C[\cdot]$ such that $C[\ell] = s$ and $C[r] = t$ for some $\ell \to r \in \mathcal{R}$. Properties like flatness extend to rules and TRSs. For example, a rule is left-flat if its left-hand side is flat; a TRS is left-flat if all its rules are.

Given a rewrite relation $\to$, we write $\to^*$ and $\leftrightarrow$ for its reflexive, transitive closure and its symmetric closure, respectively. A rewrite relation $\to$ is confluent if ${}^*\!\!\leftarrow \cdot \to^* \subseteq \to^* \cdot {}^*\!\!\leftarrow$. It is terminating if there are no infinite rewrite sequences $t_0 \to t_1 \to t_2 \to \ldots$. Two terms s and t are convertible if $s \leftrightarrow^* t$. They are joinable, denoted by $s \downarrow t$, if $s \to^* \cdot {}^*\!\!\leftarrow$.

In the complexity analysis we make use of the fact that systems of Horn clauses can be solved linear time, see Dowling and Gallier [4]. We work with propositional variables. For each variable A, there is a positive atom A and a negative atom $\neg A$. A Horn clause is a disjunction of atoms, with at most one positive atom. Horn clauses can be written as implications—the clause $\neg A \vee \neg B \vee C$ is equivalent to $A \wedge B \to C$. This implication is equivalent to the following inference rule:

$$\frac{A \quad B}{C}$$

3 Testing Confluence

We are given a finite ground TRS $\mathcal{R}_0$ over a finite signature $\mathcal{F}$. We may assume without loss of generality that $\mathcal{R}_0$ is curried, with a binary function symbol $\circ$ (representing function application) and no other non-constant function symbols. To curry an arbitrary ground TRS $\mathcal{R}$, one introduces a fresh binary function symbol $\circ$ and replaces all function applications $f(t_1, \ldots, t_n)$ by $(\ldots((f \circ t_1) \circ t_2) \ldots) \circ t_n$. The original function symbols become constants in the curried TRS. It is well-known that currying preserves (non-)confluence (e.g., [7]) and can be performed in linear time, increasing the size $\|\mathcal{R}\|$ of the TRS by a constant factor.

Furthermore we assume that $\mathcal{F}$ is minimal, i.e., only function symbols occurring in $\mathcal{R}_0$ are elements of $\mathcal{F}$. We can make this assumption because (non-)confluence is preserved under signature extension (this follows from the modularity of confluence, [11]). Let K be a countably infinite set of fresh constants (disjoint from $\mathcal{F}^{(0)}$), and let u, v, w denote elements of $\mathcal{F}^{(0)} \cup K$. We call $u \to v$ a C-rule (constant rule), and $u \circ v \to w$ a D-rule (decreasing rule).

The construction proceeds in four phases: First the TRS is flattened preserving confluence and non-confluence, then we determine its rewrite closure and congruence closure, and finally these closures are used for testing confluence of the flattened TRS.

▶ **Example 3.1.** We will decide confluence of $\mathcal{R} = \{\mathsf{a} \to \mathsf{b}, \mathsf{a} \to \mathsf{f}(\mathsf{a}), \mathsf{b} \to \mathsf{f}(\mathsf{f}(\mathsf{b}))\}$ and $\mathcal{R}' = \mathcal{R} \cup \{\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{b}))) \to \mathsf{b}\}$. We start with the curried ground TRSs $\mathcal{R}_0 = \{\mathsf{a} \to \mathsf{b}, \mathsf{a} \to \mathsf{f} \circ \mathsf{a}, \mathsf{b} \to \mathsf{f} \circ (\mathsf{f} \circ \mathsf{b})\}$ and $\mathcal{R}'_0 = \mathcal{R}_0 \cup \{\mathsf{f} \circ (\mathsf{f} \circ (\mathsf{f} \circ \mathsf{b})) \to \mathsf{b}\}$.

3.1 Flattening

First, we flatten the TRS $\mathcal{R}_0$, as follows. We start with $(\mathcal{R}_0, \varnothing)$ and exhaustively apply the rules

$$(\mathcal{R}[u \circ v], \mathcal{E}) \vdash_{\text{ext}} (\mathcal{R}[w], \mathcal{E} \cup \{u \circ v \approx w\})$$
$$(\mathcal{R}[t], \mathcal{E} \cup \{t \approx u\}) \vdash_{\text{simp}} (\mathcal{R}[u], \mathcal{E} \cup \{t \approx u\}),$$

where $w \in K$ is fresh in $\vdash_{\text{ext}}$, and $\mathcal{R}[\cdot]$ is a context of $\mathcal{R}$, i.e., a context $\ell[\cdot]$ (or $r[\cdot]$) of a side of a rule $\ell \to r$ in $\mathcal{R}$, so that $\mathcal{R}[u]$ is obtained by replacing $\ell \to r$ by $\ell[u] \to r$ (or $\ell \to r[u]$) in $\mathcal{R}$. After each $\vdash_{\text{ext}}$ step, we apply $\vdash_{\text{simp}}$ as often as possible before using $\vdash_{\text{ext}}$ again.

In the resulting pair $(\mathcal{R}, \mathcal{E})$, $\mathcal{R}$ consists solely of C-rules (since otherwise, $\vdash_{\text{ext}}$ would be applicable), and $\mathcal{E}$ consists of D-rules. Furthermore, $\to_{\mathcal{R}_0}$ is confluent, if and only if $\to_{\mathcal{E}^{\pm} \cup \mathcal{R}}$ is, since every deduction step preserves and reflects the confluence property (cf. Lemma 3.2). Because $\vdash_{\text{simp}}$ is applied eagerly, no two left-hand sides in $\mathcal{E}$ are equal, and therefore $\mathcal{E}$ is confluent (it is orthogonal). Note that as a result, every distinct subterm occurring in $\mathcal{R}$ is represented by exactly one constant from $\mathcal{F}^{(0)} \cup K$. This is similar in spirit to the Nelson-Oppen congruence closure algorithm [8].

▶ **Lemma 3.2.** *If $(\mathcal{R}, \mathcal{E}) \vdash_{\text{ext}} (\mathcal{R}', \mathcal{E}')$ or $(\mathcal{R}, \mathcal{E}) \vdash_{\text{simp}} (\mathcal{R}', \mathcal{E}')$ then $\to_{\mathcal{E}^{\pm} \cup \mathcal{R}}$ is confluent if and only if $\to_{\mathcal{E}'^{\pm} \cup \mathcal{R}'}$ is.*

Proof. The rule $\vdash_{\text{ext}}$ can be split into two steps, first adding the rule $u \circ v \approx w$ to $\mathcal{E}$ followed by applying $\vdash_{\text{simp}}$. The first step preserves confluence since any application of the new $\mathcal{E}$-rule can be undone using the corresponding $\mathcal{E}^{-}$-rule and vice versa, and since w is fresh, no rule other than $w \approx u \circ v$ can affect a subterm containing w.

For $\vdash_{\text{simp}}$, we prove that $\to_{\mathcal{E}^{\pm} \cup \mathcal{R}[t]} \subseteq \to^{*}_{\mathcal{E}^{\pm} \cup \mathcal{R}[u]}$ and $\to_{\mathcal{E}^{\pm} \cup \mathcal{R}[u]} \subseteq \to^{*}_{\mathcal{E}^{\pm} \cup \mathcal{R}[t]}$. There are two cases. (i) If a left-hand side of a rule is changed, i.e., $\ell[t] \to r \in \mathcal{R}[t]$ is changed to $\ell[u] \to r \in \mathcal{R}[u]$, then observing that $s[\ell[t]] \to_{\mathcal{E}} s[\ell[u]] \to_{\mathcal{R}[u]} s[r]$ (simulating $\to_{\ell[t] \to r}$ using rules from $\mathcal{E}^{\pm} \cup \mathcal{R}[u]$) and $s[\ell[u]] \to_{\mathcal{E}^{-}} s[\ell[t]] \to_{\mathcal{R}[t]} s[r]$ (simulating $\to_{\ell[u] \to r}$ using rules from $\mathcal{E}^{\pm} \cup \mathcal{R}[t]$) establishes the claim, since all other rules are contained in both $\mathcal{R}[u]$ and $\mathcal{R}[t]$. (ii) If a right-hand side is changed, $\ell \to r[t] \in \mathcal{R}[t]$, $\ell \to r[u] \in \mathcal{R}[u]$, then the simulations $s[\ell] \to_{\mathcal{R}[t]} s[r[t]] \to_{\mathcal{E}} s[r[u]]$ and $s[\ell] \to_{\mathcal{R}[u]} s[r[u]] \to_{\mathcal{E}^{-}} s[r[t]]$ prove the claim. ◀

Let the result of flattening be $(\mathcal{R}_1, \mathcal{E})$, over an extended signature, where $\mathcal{F}^{(0)}$ includes the fresh constants added by $\vdash_{\text{ext}}$. Flattening is straightforward to implement by a bottom-up traversal of the sides of the TRS, replacing subterms of the shape $u \circ v$ by constants, and maintaining a lookup table of which such terms have been seen before. This takes time $O(\|\mathcal{R}_0\| \log(\|\mathcal{R}_0\|))$ (the $\log(\|\mathcal{R}_0\|)$ factor accounts for the lookup table operations), and we have $\|\mathcal{E}\| = O(\|\mathcal{R}_0\|)$, $\|\mathcal{R}_1\| = O(|\mathcal{R}_0|)$, i.e., the total size of the TRSs $\mathcal{R}_1$ and $\mathcal{E}$ is at most linear in that of $\mathcal{R}_0$.

▶ **Example 3.3** (continued from Example 3.1). We introduce fresh constants fa, fb, ffb and fffb for $\mathsf{f} \circ \mathsf{a}$, $\mathsf{f} \circ \mathsf{b}$, $\mathsf{f} \circ \mathsf{fb}$ and $\mathsf{f} \circ \mathsf{ffb}$, respectively. The resulting TRSs are $(\mathcal{R}_1, \mathcal{E}) = (\{\mathsf{a} \to \mathsf{fa}, \mathsf{a} \to \mathsf{b}, \mathsf{b} \to \mathsf{ffb}\}, \{\mathsf{f}\circ\mathsf{a} \approx \mathsf{fa}, \mathsf{f}\circ\mathsf{b} \approx \mathsf{fb}, \mathsf{f}\circ\mathsf{fb} \approx \mathsf{ffb}\})$ and $(\mathcal{R}_1', \mathcal{E}') = (\mathcal{R}_1 \cup \{\mathsf{fffb} \to \mathsf{b}\}, \mathcal{E} \cup \{\mathsf{f}\circ\mathsf{ffb} \approx \mathsf{fffb})$.

3.2 Rewrite Closure

In this step, we are given a pair $(\mathcal{R}_1, \mathcal{E})$, where $\mathcal{R}_1$ is a system of C-rules and $\mathcal{E}$ is a system of D-rules. We want to obtain another pair $(\mathcal{R}_2, \mathcal{E})$, where $s \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_2} t$ iff $s \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_1} t$, such that every rewrite sequence in $(\mathcal{R}_1, \mathcal{E})$ can be transformed into a rewrite sequence in $(\mathcal{R}_2, \mathcal{E})$ of a special shape (cf. Lemma 3.4). The inference rules in Figure 1 define a relation $u \rightsquigarrow v$ on constants. We will see in a moment that $u \rightsquigarrow v$ iff $u \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_1} v$.

$$\frac{u \to v \in \mathcal{R}_1}{u \rightsquigarrow v}\ \text{base} \qquad \frac{u \in \mathcal{F}^{(0)}}{u \rightsquigarrow u}\ \text{refl} \qquad \frac{u \rightsquigarrow v \quad v \rightsquigarrow w}{u \rightsquigarrow w}\ \text{trans}$$

$$\frac{u_1 \rightsquigarrow v_1 \quad u_2 \rightsquigarrow v_2 \quad \{u_1 \circ u_2 \approx u, v_1 \circ v_2 \approx v\} \subseteq \mathcal{E}}{u \rightsquigarrow v}\ \text{comp}$$

Figure 1 Inference Rules for Rewrite Closure

The result of the rewrite closure step is $(\mathcal{R}_2, \mathcal{E})$, where $\mathcal{R}_2 = \{u \to v \mid u \rightsquigarrow v\}$.

▶ **Lemma 3.4.** *$s \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_1} t$ if and only if $s \to^*_{\mathcal{E} \cup \mathcal{R}_2} \cdot \to^*_{\mathcal{E}^- \cup \mathcal{R}_2} t$.*

Proof. Because of (base), we have $\mathcal{R}_1 \subseteq \mathcal{R}_2$, so that $\to^*_{\mathcal{E}^\pm \cup \mathcal{R}_1} \subseteq \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_2}$. On the other hand, all rules in Figure 1 are compatible with the requirement that $\to^*_{\mathcal{E}^\pm \cup \mathcal{R}_2} \subseteq \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_1}$. Therefore, the reachability relation is preserved, i.e., $\to^*_{\mathcal{E}^\pm \cup \mathcal{R}_1} = \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_2}$.

First we show that for $u, v \in \mathcal{F}^{(0)}$, $u \rightsquigarrow v$ (and therefore $u \to_{\mathcal{R}_2} v$) whenever $u \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_2} v$. Assume that we have $u \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_2} v$ but not $u \rightsquigarrow v$. Let $u = t_0 \to \cdots \to t_n = v$ be the shortest sequence of $(\mathcal{E}^\pm \cup \mathcal{R}_2)$ steps from u to v, and pick u and v such that n is minimal. If $n = 0$ then $u = v$, and $u \rightsquigarrow v$ by (refl). If $n = 1$ then $u \to v \in \mathcal{R}_2$ since $\mathcal{E}$ only contains D-rules. If $t_i \in \mathcal{F}^{(0)}$ for any $0 < i < n$, then $u \rightsquigarrow t_i \rightsquigarrow v$ by minimality of $u \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_2} v$, and $u \rightsquigarrow v$ by transitivity (trans). In the remaining case, we have $t_i = u_i \circ v_i$ for all $0 < i < n$, and hence $u_1 \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_2} u_{n-1}$ and $v_1 \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_2} v_{n-1}$ since any root step would have a constant from $\mathcal{F}^{(0)}$ as source or target. But these two rewrite sequences have length at most $n - 2$, and therefore $u_1 \rightsquigarrow u_{n-1}$ and $v_1 \rightsquigarrow v_{n-1}$, implying $u \rightsquigarrow v$ by the (comp) rule. In all cases we found that $u \rightsquigarrow v$, a contradiction.

Now let $s \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_1} t$. Then $s \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_2} t$. Assume that this rewrite sequence is not of the shape $s \to^*_{\mathcal{E} \cup \mathcal{R}_2} \cdot \to^*_{\mathcal{E}^- \cup \mathcal{R}_2} t$, but has a minimal number of inversions between $\mathcal{E}$ and $\mathcal{E}^-$ steps (an inversion is any pair of an $\mathcal{E}$ step following an $\mathcal{E}^-$ step, not necessarily directly). Then it has a subsequence of the shape $s' \to_{p,\mathcal{E}^-} s'' \to^*_{\mathcal{R}_2} t'' \to_{q,\mathcal{E}} t'$, starting with an $\mathcal{E}^-$ step at p and a final $\mathcal{E}$ step at q. The cases $p < q$ or $p > q$ are impossible, because $\mathcal{E}$ contains only D-rules and $\mathcal{R}_2$ only C-rules (applying C-rules does not change the set of positions of a term).

If $p = q$ then $s''|_{pi} \to^*_{\mathcal{R}_2} t''|_{qi}$ for $i \in \{1, 2\}$, collecting all $\mathcal{R}_2$ steps at positions below p from $s'' \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_2} t''$. By (refl) and (trans) this implies $s''|_{pi} \rightsquigarrow t''|_{qi}$ for $i \in \{1, 2\}$. Consequently, we have $s'|_p \rightsquigarrow t'|_p$ by (comp). Hence we can delete the two $\mathcal{E}^\pm$ steps and the collected $\mathcal{R}_2$ steps, and replace them by an $\mathcal{R}_2$ step using the rule $s'|_p \to t'|_p$. This decreases the number of inversions between $\mathcal{E}$ and $\mathcal{E}^-$ steps, contradicting our minimality assumption. Otherwise, if $p \parallel q$, then we can reorder the rewrite sequence $s' \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_2} t'$ as $s' \to^*_{>q,\mathcal{R}_2} \cdot \to_{q,\mathcal{E}} \cdot \to^*_{\mathcal{R}_2} \cdot \to_{p,\mathcal{E}^-} \cdot \to^*_{>p,\mathcal{R}_2} t'$, commuting mutually parallel rewrite

```
**rewrite-closure**$(n, \mathcal{E}, \mathcal{R})$: Compute rewrite closure.
(Assumes that $\mathcal{F}^{(0)} = \{1, \ldots, n\}$, which can be achieved as part of the flattening step).
1. By scanning $\mathcal{E}$ once, compute arrays $l$ and $r$ such that $l[u] = \{(v, w) \mid u \circ v \to w \in \mathcal{E}\}$
   and $r[v] = \{(u, w) \mid u \circ v \to w \in \mathcal{E}\}$.
2. Let $\mathcal{R}' = \varnothing \subseteq \{1, \ldots, n\}^2$ (represented by an array).
3. Process (refl): Call **add**$(u, u)$ for $1 \leq u \leq n$.
4. Process (base): Call **add**$(u, v)$ for $u \to v \in \mathcal{R}$.

**add**$(u, v)$: Add $u \to v$ to $\mathcal{R}'$ and process implied (trans) and (comp) rules.
1. If $u \to v \in \mathcal{R}'$, return immediately.
2. Let $\mathcal{R}' = \mathcal{R}' \cup \{u \to b\}$.
3. Process (trans): For all $w \in \{1, \ldots, n\}$,
   - if $w \to u \in \mathcal{R}'$, call **add**$(w, v)$.
   - if $v \to w \in \mathcal{R}'$, call **add**$(u, w)$.
4. Process (comp):
   - For all $(u_2, u_r) \in l[u]$ and $(v_2, v_r) \in l[v]$, if $u_2 \to v_2 \in \mathcal{R}'$, call **add**$(u_r, v_r)$.
   - For all $(u_1, u_r) \in r[u]$ and $(v_1, v_r) \in r[v]$, if $u_1 \to v_1 \in \mathcal{R}'$, call **add**$(u_r, v_r)$.
```

Figure 2 Algorithm for Rewrite Closure

steps. This reduces the number of inversions between $\mathcal{E}$ and $\mathcal{E}^-$ steps, and again we reach a contradiction. ◀

The size of $\mathcal{R}_2$ is bounded by $|\mathcal{F}^{(0)}|^2 = O(\|\mathcal{R}_0\|^2)$. We can view the inference rules in Figure 1 as a system of Horn clauses with atoms of the form $u \rightsquigarrow v$ ($u, v \in \mathcal{F}^{(0)}$). This system can be solved in time proportional to the total size of the clauses [4], finding a minimal solution for the relation $\rightsquigarrow$. There are $|\mathcal{R}_1|$ instances of (base), $|\mathcal{F}^{(0)}|$ instances of (refl), $|\mathcal{F}^{(0)}|^3$ instances of (trans) and at most $|\mathcal{F}^{(0)}|^2$ instances of (comp), noting that u_1, u_2 are determined by u and v_1, v_2 are determined by v. Therefore, we can compute $\mathcal{R}_2$ in time $O(\|R_0\|^3)$.

▶ Remark. In our implementation, we do not generate these Horn clauses explicitly. Instead, whenever we make a new inference $u \rightsquigarrow v$, we check all possible rules that involve $u \rightsquigarrow v$ as a premise. The result is a neat incremental algorithm (see Figure 2). From an abstract point of view, however, this is no different than solving the Horn clauses as stated above. This remark also applies to inference rules presented later.

▶ **Example 3.5** (continued from Example 3.3)**.** We present $\mathcal{R}_2$ and $\mathcal{R}_2'$ as tables, where non-empty entries correspond to the rules contained in each TRS. For example, fa $\to$ b $\in \mathcal{R}_2'$ but fa $\to$ b $\notin \mathcal{R}_2$. The letters indicate the inference rule used to derive the entry, while the superscripts indicate stage numbers—each inference uses only premises that have smaller stage numbers.

$\mathcal{R}_2 =$

	f	a	fa	b	fb	ffb
f	r^0					
a		r^0	b^0	b^0	t^2	t^1
fa			r^0		c^1	c^3
b				r^0		b^0
fb					r^0	
ffb						r^0

$\mathcal{R}_2' =$

	f	a	fa	b	fb	ffb	fffb
f	r^0						
a		r^0	b^0	b^0	t^2	t^1	t^3
fa			r^0	t^3	c^1	t^3	t^2
b				r^0	t^4	b^0	t^4
fb				t^2	r^0	t^2	c^1
ffb				t^4	c^3	r^0	c^3
fffb				b^0	t^4	t^1	r^0

3.3 Congruence Closure

We are also interested in the congruence closure of $(\mathcal{R}_1, \mathcal{E})$, because it allows us to decide when two terms are convertible. We calculate the congruence closure as the rewrite closure of $(\mathcal{R}_1^{\pm}, \mathcal{E})$ and call it $(\mathcal{C}, \mathcal{E})$. This step also takes $O(\|\mathcal{R}_0\|^3)$ time. By Lemma 3.4 we have

$$s \leftrightarrow^*_{\mathcal{E}^{\pm}\cup\mathcal{R}_1} t \iff s \rightarrow^*_{\mathcal{E}^{\pm}\cup\mathcal{R}_1^{\pm}} t \iff s \rightarrow^*_{\mathcal{E}\cup\mathcal{C}} \cdot \rightarrow^*_{\mathcal{E}^{-}\cup\mathcal{C}} t \iff s \downarrow_{\mathcal{E}\cup\mathcal{C}} t.$$

Note that $\mathcal{C}$ is symmetric and therefore, $\rightarrow_{\mathcal{E}^{-}\cup\mathcal{C}} = {}_{\mathcal{E}\cup\mathcal{C}}\!\leftarrow$.

▶ Remark. There are far more efficient methods for calculating the congruence closure (an almost linear time algorithm can be found in [2]), but the simple reduction to the rewrite closure is sufficient for our purposes, since the total asymptotic running time is unchanged.

▶ **Example 3.6** (continued from Example 3.5). Since $\mathcal{C}$ is an equivalence relation, we just give its equivalence classes: $[\mathsf{f}]_{\mathcal{C}} = \{\mathsf{f}\}$ and $[\mathsf{a}]_{\mathcal{C}} = \{\mathsf{a}, \mathsf{fa}, \mathsf{b}, \mathsf{fb}, \mathsf{ffb}\}$. For $\mathcal{C}'$, we obtain $[\mathsf{f}]_{\mathcal{C}'} = [\mathsf{f}]_{\mathcal{C}}$ and $[\mathsf{a}]_{\mathcal{C}'} = [\mathsf{a}]_{\mathcal{C}} \cup \{\mathsf{fffb}\}$. Note that $\mathcal{C}$ and $\mathcal{C}'$ are the symmetric, transitive closures of $\mathcal{R}_2$ and $\mathcal{R}_2'$, respectively. This holds in general.

3.4 Confluence Conditions

So far we have flattened the TRS $\mathcal{R}_0$ and computed its rewrite and congruence closures, enabling us to check reachability and convertibility of any given terms efficiently. In this section we use these tools to decide confluence of $\mathcal{R}_0$.

We closely follow the approach in [10] and [6], which is based on the analysis of two convertible terms s, t and their normal forms with respect to a system of so-called *forward rules* of a rewrite closure. In our approach, these correspond to the system $\mathcal{E} \cup \mathcal{R}_2$. However, $\rightarrow_{\mathcal{E}\cup\mathcal{R}_2}$ is typically non-terminating, and we cannot use this idea directly. This problem is easy to overcome though. We define $\mathcal{A};\mathcal{B} = \{\ell \rightarrow r \mid \ell \rightarrow m \in \mathcal{A} \text{ and } m \rightarrow r \in \mathcal{B}\}$ and $\rightarrow_{\mathcal{A}/\mathcal{B}} = \rightarrow^*_{\mathcal{B}} \cdot \rightarrow_{\mathcal{A}} \cdot \rightarrow^*_{\mathcal{B}}$. Note that $\rightarrow_{\mathcal{E}/\mathcal{R}_2}$ is terminating. We will use $\rightarrow_{\mathcal{E}/\mathcal{R}_2}$ in place of the forward reduction. This choice is justified by Lemma 3.8 below. We will abuse notation slightly and speak of $\mathcal{E}/\mathcal{R}_2$ normal forms.

▶ **Lemma 3.7.** *Let $\mathcal{S}$ be a transitive, reflexive (as a relation) set of C-rules and $\mathcal{E}$ a set of D-rules. Then $\rightarrow^*_{\mathcal{E}\cup\mathcal{S}} = \rightarrow^*_{\mathcal{S}} \cdot \rightarrow^*_{\mathcal{E};\mathcal{S}}$ and $\rightarrow^*_{\mathcal{E}^{-}\cup\mathcal{S}} = \rightarrow^*_{\mathcal{S};\mathcal{E}^{-}} \cdot \rightarrow^*_{\mathcal{S}}$.*

Proof. We first show that $\rightarrow^*_{\mathcal{E}\cup\mathcal{S}} = \rightarrow^*_{\mathcal{S}} \cdot \rightarrow^*_{\mathcal{E};\mathcal{S}}$. Start with a rewrite sequence $s \rightarrow^*_{\mathcal{E}\cup\mathcal{S}} t$. Whenever an $\mathcal{S}$ step is followed by another $\mathcal{S}$ step at the same position, we can combine them using transitivity of $\mathcal{S}$. Note that since $\mathcal{E}$ only contains D-rules, no intermediate $\mathcal{E}$ step can overlap with either of the $\mathcal{S}$ steps. Once there are no more $\mathcal{S}$ steps that can be combined this way, we replace each $\mathcal{E}$ step that is followed by an $\mathcal{S}$ step at the same position by the corresponding $\mathcal{E};\mathcal{S}$ step. If there is no following $\mathcal{S}$ step, we add an identity $\mathcal{S}$ step (which exists by reflexivity of $\mathcal{S}$) first. It is easy to verify that the final rewrite sequence is of the desired shape.

For $\rightarrow^*_{\mathcal{E}^{-}\cup\mathcal{S}} = \rightarrow^*_{\mathcal{S};\mathcal{E}^{-}} \cdot \rightarrow^*_{\mathcal{S}}$ it suffices to note that by reversing the rewrite sequences this is equivalent to $\rightarrow^*_{\mathcal{E}\cup\mathcal{S}^{-}} = \rightarrow^*_{\mathcal{S}^{-}} \cdot \rightarrow^*_{\mathcal{E};\mathcal{S}^{-}}$. Since $\mathcal{S}^{-}$ is transitive and reflexive if $\mathcal{S}$ is, the claim reduces to the previous one. ◀

▶ **Lemma 3.8.**

1. *If $s \rightarrow^*_{\mathcal{E}^{\pm}\cup\mathcal{R}_1} t$ then $s \rightarrow^*_{\mathcal{E}/\mathcal{R}_2} \cdot \rightarrow^*_{\mathcal{R}_2;\mathcal{E}^{-}} \cdot \rightarrow^*_{\mathcal{R}_2} t$.*
2. *If $s \leftrightarrow^*_{\mathcal{E}^{\pm}\cup\mathcal{R}_1} t$ then $s \rightarrow^*_{\mathcal{C}} \cdot \rightarrow^*_{\mathcal{E};\mathcal{C}} \cdot {}_{\mathcal{E};\mathcal{C}}\!\!\overset{*}{\leftarrow} \cdot {}_{\mathcal{C}}\!\!\overset{*}{\leftarrow} t$.*

Proof. 1. Assume that $s \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_1} t$. By Lemma 3.4, this is equivalent to $s \to^*_{\mathcal{E} \cup \mathcal{R}_2} \cdot \to^*_{\mathcal{E}^- \cup \mathcal{R}_2} t$, or $s \to^*_{\mathcal{E}/\mathcal{R}_2} \cdot \to^*_{\mathcal{E}^- \cup \mathcal{R}_2} t$, which according to Lemma 3.7 is equivalent to $s \to^*_{\mathcal{E}/\mathcal{R}_2} \cdot \to^*_{\mathcal{R}_2;\mathcal{E}^-} \cdot \to^*_{\mathcal{R}_2} t$, noting that $\mathcal{R}_2^-$ is both reflexive and transitive by construction.

2. Assume that $s \leftrightarrow^*_{\mathcal{E}^\pm \cup \mathcal{R}_1} t$, i.e., $s \to_{\mathcal{E}^\pm \cup \mathcal{R}_1^\pm} t$. Again by Lemma 3.4, this is equivalent to $s \to^*_{\mathcal{E} \cup \mathcal{C}} \cdot {}_{\mathcal{E} \cup \mathcal{C}}{}^*\!\!\leftarrow t$. Since $\mathcal{C}$ is reflexive and transitive, the claim follows from Lemma 3.7. ◀

Let us assume that $\mathcal{R}_1 \cup \mathcal{E}^\pm$ is confluent, and that we have two convertible terms s and t. There are corresponding $\mathcal{E}/\mathcal{R}_2$ normal forms s' and t' for s and t, respectively. Now s' and t' are convertible, so that by Lemma 3.8(2), for some term r,

$$s' \to^*_{\mathcal{C}} \cdot \to^*_{\mathcal{E};\mathcal{C}} r \; {}_{\mathcal{E};\mathcal{C}}{}^*\!\!\leftarrow \cdot \; {}_{\mathcal{C}}{}^*\!\!\leftarrow t'. \tag{1}$$

Furthermore, by confluence and Lemma 3.8(1), noting that the choice of s' and t' forces the $\to^*_{\mathcal{E}/\mathcal{R}_2}$ sequences to be empty, it follows that for their common reduct r',

$$s' \to^*_{\mathcal{R}_2;\mathcal{E}^-} \cdot \to^*_{\mathcal{R}_2} r' \; {}_{\mathcal{R}_2}{}^*\!\!\leftarrow \cdot \; {}_{\mathcal{R}_2;\mathcal{E}^-}{}^*\!\!\leftarrow t'. \tag{2}$$

To capture the conditions on s' and t' (which are $\mathcal{E}/\mathcal{R}_2$ normal forms), we adapt the notion of *top-stabilizable* terms and constants from [6] to our purposes.

▶ **Definition 3.9.** A term $u \circ v$ with $u, v \in \mathcal{F}^{(0)}$ is *top-stabilizable* if there exists an $\mathcal{E}/\mathcal{R}_2$ normal form s such that $s \to^*_{\mathcal{C}} \cdot \to^*_{\mathcal{E};\mathcal{C}} u \circ v$. A constant $u \in \mathcal{F}^{(0)}$ is *top-stabilizable* if there exist $v, w \in \mathcal{F}^{(0)}$ such that $u \to_{\mathcal{C};\mathcal{E}^-} v \circ w$ and $v \circ w$ is top-stabilizable.

The equations (1,2) define two rewrite sequences from r to r' that consist solely of C- and inverse D-steps (note that we consider the rewrite sequences from (1) in reverse). This means that no rewrite step occurs below a preceding rewrite step. In fact all rewrite steps modify a leaf of a term. Therefore we may assume without loss of generality that $r \in \mathcal{F}^{(0)}$. Looking at the surrounding rewrite steps in equation (1), we distinguish three cases depending on whether the sequence of $\mathcal{E};\mathcal{C}$ steps is empty or not.

1. $s' \to^*_{\mathcal{C} \cup \mathcal{E}} s_1 \circ s_2 \to_{\mathcal{E};\mathcal{C}} r \; {}_{\mathcal{E};\mathcal{C}}\!\!\leftarrow t_1 \circ t_2 \; {}_{\mathcal{C} \cup \mathcal{E}}{}^*\!\!\leftarrow t'$. In this case $s_1 \circ s_2$, $t_1 \circ t_2$ must be top-stabilizable. Furthermore, for $i \in \{1,2\}$, the terms s_i and t_i are convertible via r_i, so that $s_i \downarrow_{\mathcal{C} \cup \mathcal{E}} t_i$ by Lemma 3.8.
2. $s' \to^*_{\mathcal{C} \cup \mathcal{E}} s_1 \circ s_2 \to_{\mathcal{E};\mathcal{C}} t' \in \mathcal{F}^{(0)}$. (Note that we use the fact that $\mathcal{C}$ is an equivalence relation: $s_1 \circ s_2 \to_{\mathcal{E};\mathcal{C}} \cdot \; {}_{\mathcal{C}}{}^*\!\!\leftarrow t'$ implies $s_1 \circ s_2 \to_{\mathcal{E};\mathcal{C}} t'$ if $t' \in \mathcal{F}^{(0)}$.) Then there must be $t_1, t_2 \in \mathcal{F}^{(0)}$ such that $t' \to_{\mathcal{R}_2;\mathcal{E}^-} t_1 \circ t_2$, and $s_i \downarrow_{\mathcal{C} \cup \mathcal{E}} t_i$ for $i \in \{1,2\}$. This case also covers $s' \; {}_{\mathcal{E};\mathcal{C}}\!\!\leftarrow t_1 \circ t_2 \; {}_{\mathcal{C} \cup \mathcal{E}}{}^*\!\!\leftarrow t'$ by symmetry.
3. $\mathcal{F}^{(0)} \ni s' \to_{\mathcal{C}} t' \in \mathcal{F}^{(0)}$. Then $s' \downarrow_{\mathcal{E}^- \cup \mathcal{R}_2} t'$, with common reduct r'.

Hence we have found the following necessary conditions for confluence of $\mathcal{R}_1 \cup \mathcal{E}^\pm$:

▶ **Definition 3.10.** The *confluence conditions* for confluence of $\mathcal{R}_2 \cup \mathcal{E}^\pm$ are as follows.
1. If $s_1 \circ s_2$ and $t_1 \circ t_2$ are top-stabilizable for constants $s_1, s_2, t_1, t_2 \in \mathcal{F}^{(0)}$ such that $s_1 \circ s_2 \to_{\mathcal{E};\mathcal{C}} r \; {}_{\mathcal{E};\mathcal{C}}\!\!\leftarrow t_1 \circ t_2$ then $s_i \downarrow_{\mathcal{C} \cup \mathcal{E}} t_i$ for $i \in \{1,2\}$.
2. If $s_1 \circ s_2 \to_{\mathcal{E};\mathcal{C}} t'$ for $s_1, s_2, t' \in \mathcal{F}^{(0)}$ and top-stabilizable $s_1 \circ s_2$, then there must be $t_1, t_2 \in \mathcal{F}^{(0)}$ such that $t' \to_{\mathcal{R}_2;\mathcal{E}^-} t_1 \circ t_2$, and $s_i \downarrow_{\mathcal{C} \cup \mathcal{E}} t_i$ for $i \in \{1,2\}$.
3. If $\mathcal{F}^{(0)} \ni s' \to_{\mathcal{C}} t' \in \mathcal{F}^{(0)}$ then $s' \downarrow_{\mathcal{E}^- \cup \mathcal{R}_2} t'$.

▶ **Lemma 3.11.** *The confluence conditions are necessary and sufficient for confluence of* $\mathcal{R}_1 \cup \mathcal{E}^\pm$.

Proof. Necessity has already been shown above. For sufficiency, assume that the confluence conditions are satisfied and there are convertible terms s and t with no common reduct. Then any corresponding $\mathcal{E}/\mathcal{R}_2$ normal forms do not have a common reduct either. Let s' and t' be convertible $\mathcal{E}/\mathcal{R}_2$ normal forms with no common reduct such that $|s'| + |t'|$ is minimal. Recall that $\to^*_{\mathcal{E}^\pm \cup \mathcal{R}_1} = \to^*_{\mathcal{E}^\pm \cup \mathcal{R}_2}$ so that $\mathcal{R}_1 \cup \mathcal{E}^\pm$ joinability and $\mathcal{R}_2 \cup \mathcal{E}^\pm$ joinability coincide. The same holds for convertibility. We will simply use the terms "joinable" and "convertible" for both $\mathcal{R}_1 \cup \mathcal{E}^\pm$ and $\mathcal{R}_2 \cup \mathcal{E}^\pm$. We distinguish three cases.

1. If $s', t' \in \mathcal{F}^{(0)}$, then by Lemma 3.8(2), $s' \to_{\mathcal{C}} t'$ (since s', t' are $\mathcal{E};\mathcal{C}$ normal forms and $\mathcal{C}$ is an equivalence relation) and we obtain a joining sequence from the third confluence condition, contradicting the non-joinability of s' and t'.
2. If $s' = s'_1 \circ s'_2 \notin \mathcal{F}^{(0)}$ and $t' \in \mathcal{F}^{(0)}$, then by Lemma 3.8(2) there is a rewrite sequence $s' \to^*_{\mathcal{C}\cup\mathcal{E}} s_1 \circ s_2 \to_{\mathcal{E};\mathcal{C}} t'$ (again using that t' is an $\mathcal{E};\mathcal{C}$ normal form and that $\mathcal{C}$ is an equivalence relation). By the second confluence condition we obtain a term $t_1 \circ t_2$ such that $t' \to_{\mathcal{R}_2;\mathcal{E}^-} t_1 \circ t_2$, and t_i and s_i are convertible for $i \in \{1,2\}$. Therefore, t_1 and s'_1 are convertible. Furthermore, since $|t_1| + |s'_1| < |t'| + |s'|$, this implies that t_1 and s'_1 are joinable. Analogously, t_2 and s'_2 are also joinable, and therefore s' is joinable with $t_1 \circ t_2 \;{}_{\mathcal{R}_2;\mathcal{E}^-}\!\leftarrow t'$, contradicting our assumptions.
   The case that $s' \in \mathcal{F}^{(0)}$ and $t' \notin \mathcal{F}^{(0)}$ is handled symmetrically.
3. If $s' = s'_1 \circ s'_2 \notin \mathcal{F}^{(0)}$ and $t' = t'_1 \circ t'_2 \notin \mathcal{F}^{(0)}$, then by Lemma 3.8(2), $s' \to^*_{\mathcal{E}\cup\mathcal{C}} r \;{}_{\mathcal{E}\cup\mathcal{C}}^{*}\!\leftarrow t'$. If $r = r_1 \circ r_2$ is not a constant, then s'_1 and t'_1 are convertible via r_1 and likewise s'_2 and t'_2 are convertible via r_2. However, one of these pairs cannot be joinable, and we obtain a smaller counterexample to confluence, a contradiction. Therefore, r must be a constant. Using Lemma 3.8(2) we obtain a rewrite proof $s' \to^*_{\mathcal{C}\cup\mathcal{E}} s_1 \circ s_2 \to_{\mathcal{E};\mathcal{C}} r \;{}_{\mathcal{E};\mathcal{C}}\!\leftarrow t_1 \circ t_2 \;{}_{\mathcal{C}\cup\mathcal{E}}^{*}\!\leftarrow t'$. From the first confluence condition, we conclude that s_1 and t_1 are convertible and therefore also s'_1 and t'_1. By minimality of $|s'| + |t'|$, s'_1 and t'_1 must be joinable. Likewise, s'_2 and t'_2 must also be joinable, from which we conclude that $s' = s'_1 \circ s'_2$ and $t' = t'_1 \circ t'_2$ are joinable as well, a contradiction.

This completes the proof. ◀

3.5 Computation of Confluence Conditions

The computation consists of two major steps: First we compute all top-stabilizable constants and terms of the form $u \circ v$. Then we check the three confluence conditions.

In order to compute the top-stabilizable constants and terms, we first need to find the $\mathcal{E}/\mathcal{R}_2$ normal forms of the shape $u \circ v$—denoted by $\mathsf{NF}(u \circ v)$. We can compute the complement of that set, i.e., the $\mathcal{E}/\mathcal{R}_2$ reducible terms of that shape using the following inference rules.

$$\frac{u \circ v \approx w \in \mathcal{E}}{\neg\mathsf{NF}(u \circ v)}\ \text{base} \qquad \frac{\{u_1 \to v_1, u_2 \to v_2\} \subseteq \mathcal{R}_2 \quad \neg\mathsf{NF}(v_1 \circ v_2)}{\neg\mathsf{NF}(u_1 \circ u_2)}\ \text{comp}$$

To obtain a cubic time algorithm, note that thanks to transitivity of $\mathcal{R}_2$, inferences made by (comp) need not be processed—if $\neg\mathsf{NF}(w_1 \circ w_2)$ implies $\neg\mathsf{NF}(v_1 \circ v_2)$ by (comp) and $\neg\mathsf{NF}(v_1 \circ v_2)$ implies $\neg\mathsf{NF}(u_1 \circ u_2)$ by (comp) then $\neg\mathsf{NF}(w_1 \circ w_2)$ implies $\neg\mathsf{NF}(u_1 \circ u_2)$ by (comp) as well. Therefore we simply consider each $\mathcal{E}$ rule (there are $O(\|\mathcal{R}_0\|)$ of these) in turn, and then make the corresponding inferences by the (comp) rule in $O(\|\mathcal{R}_0\|^2)$ time, for a total of $O(\|\mathcal{R}_0\|^3)$.

Let us turn to top-stabilizable terms and constants now. First note that any constant is an $\mathcal{E}/\mathcal{R}_2$ normal form. The top-stabilizable constants and terms can be found using another incremental computation. Every $\mathcal{E}/\mathcal{R}_2$ normal form is top-stabilizable. If $u \circ v$ is top-stabilizable and $u \circ v \to_{\mathcal{E}/\mathcal{C}} w$, then w is a top-stabilizable constant, and $u' \circ v$ and $u \circ v'$ are top-stabilizable terms whenever $u \to_{\mathcal{C}} u'$, $v \to_{\mathcal{C}} v'$. For any top-stabilizable constant w, $w \circ v$, $u \circ w$ for constants u, v are also top-stabilizable. Consequently, we obtain the following inference rules, where $\mathsf{TS}(u)$ and $\mathsf{TS}(v \circ w)$ assert that u and $v \circ w$ are top-stabilizable, respectively, and $(i, i') \in \{(1,2),(2,1)\}$.

$$\frac{u_1 \circ u_2 \in \mathsf{NF}(\mathcal{E}/\mathcal{R}_2)}{\mathsf{TS}(u_1 \circ u_2)}\ \mathrm{nf} \qquad \frac{u_1 \circ u_2 \approx u \in \mathcal{E} \quad \mathsf{TS}(u_1 \circ u_2)}{\mathsf{TS}(u)}\ \mathrm{ts}_0 \qquad \frac{\mathsf{TS}(u_i)}{\mathsf{TS}(u_1 \circ u_2)}\ \mathrm{ts}_i$$

$$\frac{u \to v \in \mathcal{C} \quad \mathsf{TS}(v)}{\mathsf{TS}(u)}\ \mathrm{comp}_0 \qquad \frac{u_i \to v_i \in \mathcal{C} \quad u_{i'} = v_{i'} \quad \mathsf{TS}(v_1 \circ v_2)}{\mathsf{TS}(u_1 \circ u_2)}\ \mathrm{comp}_i$$

There are $O(\|\mathcal{R}_0\|^2)$ instances of (nf), ($\mathrm{ts}_{\{1,2\}}$) and (comp_0), and $O(\|\mathcal{R}_0\|^3)$ instances of (ts_0) and ($\mathrm{comp}_{\{1,2\}}$). Again these inference rules have the shape of Horn clauses and can be processed in time proportional to their total size, which is $O(\|\mathcal{R}_0\|^3)$.

▶ **Example 3.12** (continued from Example 3.6). For $\mathcal{R}$ we have $\neg\mathsf{NF} = \{\mathsf{f} \circ \mathsf{b}, \mathsf{f} \circ \mathsf{fa}, \mathsf{f} \circ \mathsf{fb}, \mathsf{f} \circ \mathsf{a}\}$. Indeed $\mathsf{f} \circ \mathsf{ffb}$ is an $\mathcal{E}/\mathcal{R}_2$ normal form since using $\mathcal{R}_2$ it can only be rewritten to itself and it is not the left-hand side of any $\mathcal{E}$ rule. On the other hand, for $\mathcal{R}'$ we obtain $\neg\mathsf{NF}' = \neg\mathsf{NF} \cup \{\mathsf{f} \circ \mathsf{ffb}, \mathsf{f} \circ \mathsf{fffb}\}$. Note that normal forms also include terms like $\mathsf{f} \circ \mathsf{f}$ or $\mathsf{fa} \circ \mathsf{a}$ that have no correspondence in the original TRS.

In the $\mathcal{R}$ case, all terms of the form $u \circ v$ are top-stabilizable and so are all constants except for f. For $\mathcal{R}'$, only the normal forms are top-stabilizable.

With this pre-computation done, checking the confluence conditions becomes a straightforward matter. The only tricky part is checking joinability of constants in the third condition. This relation can be computed in a way strikingly similar to the rewrite closure from Section 3.2, using the following inference rules for computing $\downarrow = \downarrow_{\mathcal{E}^- \cup \mathcal{R}_2}$ on constants:

$$\frac{u \in \mathcal{F}^{(0)}}{u \downarrow u}\ \mathrm{refl} \qquad \frac{\{u_1 \circ u_2 \approx u, v_1 \circ v_2 \approx v\} \subseteq \mathcal{E} \quad u_1 \downarrow v_1 \quad u_2 \downarrow v_2}{u \downarrow v}\ \mathrm{comp}$$

$$\frac{u \to v \in \mathcal{R}_2 \quad v \downarrow w}{u \downarrow w}\ \mathrm{trans}_l \qquad \frac{u \downarrow v \quad w \to v \in \mathcal{R}_2}{u \downarrow w}\ \mathrm{trans}_r$$

As with the previous inference rules, this is a system of Horn clauses. There are $O(\|\mathcal{E}\|^2) = O(\|\mathcal{R}_0\|^2)$ instances of (comp), $O(\|\mathcal{R}_0\|)$ instances of (refl) and $O(\|\mathcal{R}_0\|^3)$ instances of ($\mathrm{trans}_{l,r}$). Therefore, computing $\downarrow$ can be done in $O(\|\mathcal{R}_0\|^3)$ time.

▶ **Example 3.13** (continued from Example 3.12). The joinability relations $\mathcal{R}$ and $\mathcal{R}'$ are given below. As in Example 3.5, the letters and superscripts indicate the rule being used to derive the entry and computation stage.

$\downarrow =$

	f	a	fa	b	fb	ffb
f	r^0					
a		r^0	t_l^1	t_l^1	t_l^1	t_l^1
fa		t_r^1	r^0	t_l^1	t_l^1	t_l^1
b		t_r^1	t_r^1	r^0		t_l^1
fb		t_r^1	t_r^1		r^0	
ffb		t_r^1	t_r^1	t_r^1		r^0

$\downarrow' =$

	f	a	fa	b	fb	ffb	fffb
f	r^0						
a		r^0	t_l^1	t_l^1	t_l^1	t_l^1	t_l^1
fa		t_r^1	r^0	t_l^1	c^2	c^2	c^2
b		t_r^1	t_r^1	r^0	t_r^1	c^3	c^3
fb		t_r^1	c^2	t_l^1	r^0	c^2	c^4
ffb		t_r^1	c^2	c^3	c^2	r^0	c^3
fffb		t_r^1	c^2	c^3	c^4	c^3	r^0

It is now easy to verify that $\mathcal{R}$ violates the third confluence condition ($\mathsf{fb} \to_{\mathcal{C}} \mathsf{ffb}$ but not $\mathsf{fb} \downarrow \mathsf{ffb}$), and therefore is not confluent. The other two confluence conditions are satisfied. $\mathcal{R}'$, on the other hand, satisfies all confluence conditions and is, therefore, confluent.

Table 1 Confluence of Ground Cops.

Cop	21	33	34	38	39	40	80	81	84	114	115	116
CR	×	✓	✓	×	×	✓	×	✓	✓	✓	✓	✓

Table 2 Runtimes for $\mathcal{R}_n$.

system	$\mathcal{R}_{100}$	$\mathcal{R}_{200}$	$\mathcal{R}_{400}$	$\mathcal{R}_{800}$	$\mathcal{R}_{1600}$	$\mathcal{R}_{3200}$
time (s)	0.2 (×)	0.2 (×)	1.3 (×)	19.2 (×)	254.3 (×)	2321 (×)
system	$\mathcal{R}_{101}$	$\mathcal{R}_{201}$	$\mathcal{R}_{401}$	$\mathcal{R}_{801}$	$\mathcal{R}_{1601}$	$\mathcal{R}_{3201}$
time (s)	0.2 (✓)	0.2 (✓)	2.3 (✓)	30.1 (✓)	427.4 (✓)	3919 (✓)

Putting everything together, we obtain the following theorem.

▶ **Theorem 3.14.** *The confluence of a ground TRS $\mathcal{R}$ can be decided in cubic time.*

Proof. Let $n = \|\mathcal{R}\|$. We follow the process outlined above. First we curry $\mathcal{R}$ in linear time, obtaining $\mathcal{R}_0$ with $\|\mathcal{R}_0\| = O(n)$. Then we flatten $\mathcal{R}_0$, obtaining $(\mathcal{R}_1, \mathcal{E})$ with $\|\mathcal{E}\| = O(n)$ and $\|\mathcal{R}_1\| = O(n)$ in time $O(n \log(n))$. In the next step we compute the rewrite and congruence closures $(\mathcal{R}_2, \mathcal{E})$ and $(\mathcal{C}, \mathcal{E})$ of $(\mathcal{R}_1, \mathcal{E})$ in $O(n^3)$ time. Afterwards, we compute the $\mathcal{E}/\mathcal{R}_2$ normal forms $\mathsf{NF}(- \circ -)$, which as seen above takes $O(n^3)$ time. We then compute $\mathsf{TS}(-)$, $\mathsf{TS}(-,-)$ and $\downarrow_{\mathcal{E}^- \cup \mathcal{R}_2}$ in $O(n^3)$ time. Finally we check the three confluence conditions. For the first condition, we check each of the $O(n^2)$ pairs of rules $s_1 \circ s_2 \to_\mathcal{E} u, t_1 \circ t_2 \to_\mathcal{E} v$ with $u \to_\mathcal{C} v$. For the second condition, we consider the $O(n^3)$ triples such that $s_1 \circ s_2 \to_\mathcal{E} u \to_\mathcal{C} t' \to_{\mathcal{R}_2} v \;{}_\mathcal{E}\!\leftarrow t_1 \circ t_2$. For the third condition we check all $O(n^2)$ pairs $s' \to_\mathcal{C} t'$. All these steps can be accomplished in $O(n^3)$ time. ◀

4 Experiments

We have implemented the above algorithm in the confluence tool CSI[2] [12], and tested it on the ground confluence problems from the Cops database.[3] The results are displayed in Table 1. There are no runtimes given because they are all negligible. Note though that even before implementing ground confluence in CSI, the tool could handle all these problems. The runtime improved from 14s to 3s for checking all the TRSs. In order to obtain runtime measurements, we considered the family of TRSs $\mathcal{R}_n = \mathcal{R} \cup \{\mathsf{f}^n(\mathsf{b}) \to \mathsf{b}\}$ extending $\mathcal{R}$ from Example 3.1. One can easily argue that the system $\mathcal{R}_n$ is confluent if and only if n is odd. (Since $\mathcal{R}$ is a subsystem, all terms are convertible. However, $\mathsf{f}(\mathsf{b})$ and b are only joinable if n is odd—otherwise the parity of k in the reducts $\mathsf{f}^k(\mathsf{b})$ is invariant.) The runtimes for various n are given in Table 2. ACP[4] [1] and Saigawa[5] fail on all these systems. The numbers from Table 2 do not agree well with the proven complexity bound. This is due to cache effects—as the input size increases, the intermediate arrays outgrow the first and second level caches. Note that for the last two columns, the factor is very close to 8, finally meeting expectations. The difference between odd and even n can be explained by the different size of the rewrite closures. All measurements were done on a 2.67GHz Intel i7-620M computer with 4GB RAM using a single core.

[2] `http://cl-informatik.uibk.ac.at/software/csi/`
[3] `http://coco.nue.riec.tohoku.ac.jp/cops/`
[4] version 0.20, `http://www.nue.riec.tohoku.ac.jp/tools/acp/`
[5] version 1.2, `http://www.jaist.ac.jp/project/saigawa/`

5 Conclusion

We have described an efficient algorithm for deciding the confluence of ground TRSs. In our opinion, this is a worthwhile addition to an automated confluence checker, since other methods fail on relatively simple ground TRSs. In fact, ACP can not handle either TRS from Example 3.1, and neither can Saigawa. Before adding the ground TRS code, CSI could not disprove confluence of $\mathcal{R}$, but it was able to prove confluence of $\mathcal{R}'$. It still failed on a close relative of $\mathcal{R}'$, namely the confluent ground TRS $\mathcal{R}_5 = \mathcal{R} \cup \{\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{b}))))) \to \mathsf{b}\}$.

A natural question is whether we can improve the bounds for the other known classes of TRSs with fixed maximum arity that have a known polynomial complexity for deciding complexity, foremost the class of shallow, left-linear TRSs. Our main improvement over [10] is the limitation to C-rules in the rewrite closure, effectively constraining the considered rules to relations between subterms of the original curried TRS. This does no longer work once we have variables in rules. Therefore, at present, we do not know how to improve the other results.

Acknowledgements The author is grateful to Aart Middeldorp, Harald Zankl and Sarah Winkler for comments on a preliminary draft of this paper, and to the anonymous reviewers for advice that helped to improve the presentation of the final version.

References

1 T. Aoto, J. Yoshida, and Y. Toyama. Proving confluence of term rewriting systems automatically. In *Proc. 20th RTA*, volume 5595 of *LNCS*, pages 93–102, 2009.
2 F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
3 H. Comon, G. Godoy, and R. Nieuwenhuis. The confluence of ground term rewrite systems is decidable in polynomial time. In *Proc. 42nd FOCS*, pages 298–307, 2001.
4 W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.
5 G. Godoy, A. Tiwari, and R. Verma. On the confluence of linear shallow term rewrite systems. In *Proc. 20th STACS*, volume 2607 of *LNCS*, pages 85–96, 2003.
6 G. Godoy, A. Tiwari, and R. Verma. Deciding confluence of certain term rewriting systems in polynomial time. *Annals of Pure and Applied Logic*, 130(1-3):33–59, 2004.
7 S. Kahrs. Confluence of curried term-rewriting systems. *JSC*, 19(6):601–623, 1995.
8 G. Nelson and D.C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980.
9 D. Plaisted. Polynomial time termination and constraint satisfaction tests. In *Proc. 5th RTA*, volume 690 of *LNCS*, pages 405–420, 1993.
10 A. Tiwari. Deciding confluence of certain term rewriting systems in polynomial time. In *Proc. 17th LICS*, pages 447–457, 2002.
11 Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.
12 H. Zankl, B. Felgenhauer, and A. Middeldorp. CSI – A confluence tool. In *Proc. 23rd CADE*, volume 6803 of *LNCS (LNAI)*, pages 499–505, 2011.

Polynomial Interpretations for Higher-Order Rewriting *

Carsten Fuhs[1] and Cynthia Kop[2]

1 **University College London, Gower Street, London WC1E 6BT, UK**
2 **Vrije Universiteit, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands**

Abstract

The termination method of weakly monotonic algebras, which has been defined for higher-order rewriting in the HRS formalism, offers a lot of power, but has seen little use in recent years. We adapt and extend this method to the alternative formalism of *algebraic functional systems*, where the simply-typed λ-calculus is combined with algebraic reduction. Using this theory, we define *higher-order polynomial interpretations*, and show how the implementation challenges of this technique can be tackled. A full implementation is provided in the termination tool WANDA.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems – Decision problems

Keywords and phrases higher-order rewriting, termination, polynomial interpretations, weakly monotonic algebras, automation

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.176

Category Regular Research Paper

1 Introduction

One of the most prominent techniques in termination proofs for first-order term rewriting systems (TRSs) is the use of *polynomial interpretations*. In this method, which dates back to the seventies [24], terms are mapped to polynomials over (e.g.) $\mathbb{N}$. The method is quite intuitive, since a TRS is usually written with a meaning for the function symbols in mind, which can often be modeled by the interpretation. In addition, it has been implemented in various automatic tools, such as AProVE [14], $\mathsf{T_TT_2}$ [22] and Jambox [8]. Polynomial interpretations are an instance of the *monotonic algebra* approach [9] which also includes for instance *matrix interpretations*. They are used both on their own, and in combination with *dependency pair* approaches [1].

In the higher-order world, monotonic algebras were among the first termination methods to be defined, appearing as early as 1994 [26]; an in-depth study is done in van de Pol's 1996 PhD thesis [27]. Surprisingly, the method has been almost entirely absent from the literature ever since. This is despite a lot of interest in higher-order rewriting, witnessed not only by a fair number of publications, but also by the recent participation of higher-order tools in the *annual Termination Competition* [30]. Since the addition of a higher-order category, two tools have participated: THOR [4], by Borralleras and Rubio, and WANDA [18], by the second author of this paper. So far, neither tool has implemented weakly monotonic algebras.

* This research is supported by the Netherlands Organisation for Scientific Research (NWO-EW) under grant 612.000.629 (HOT).

In this paper we aim to counteract this situation, by both studying the class of polynomial interpretations in the natural numbers, and implementing the resulting technique in the termination tool WANDA. Van de Pol did not consider automation of his method (there was less focus on automation at the time), but there are now years of experience of the first-order world to build on; we will lift the parametric first-order approach [6], and make some necessary adaptations to cater for the presence of higher-order variables.

Paper Setup. Section 2 discusses preliminaries: *Algebraic Functional Systems*, the higher-order formalism we consider, reduction pairs and weakly monotonic algebras for typed λ-terms. In Section 3 we extend these definitions to AFSs, and define a general termination method. Section 4 defines the class of *higher-order polynomials*, and in Section 5 we show how suitable polynomial interpretations can be found automatically. Experiments with this implementation are presented in Section 6, and an overview and ideas for future work are given in Section 7.

The main contribution of this paper are the techniques for automation, discussed in Section 6. For simplicity of the code, these techniques are limited to the (very common) class of second-order AFSs, although extensions to systems of a higher order are possible. As far as we know, this is the first implementation of higher-order polynomial interpretations.

An extended version of this paper with more elaborate proofs is available at [13].

2 Background

2.1 Algebraic Functional Systems

We consider algebraic higher-order rewriting as defined by Jouannaud and Okada, also called *Algebraic Functional Systems (AFSs)* [16]. This formalism combines the simply-typed λ-calculus with algebraic reduction, and appears in papers on e.g. HORPO [17], MHOSPO [5] and dependency pairs [20]; it is also the formalism in the higher-order category of the annual termination competition. We follow roughly the definitions in [29, Ch. 11.2.3], as recalled below.

Types and Terms. The set of *simple types* (or just *types*) is generated from a given set $\mathcal{B}$ of *base types* and the binary, right-associative type constructor $\Rightarrow$; types are denoted by σ, τ and base types by ι, κ. A type with at least one occurrence of $\Rightarrow$ is called a *functional type*. A *type declaration* is an expression of the form $[\sigma_1 \times \ldots \times \sigma_n] \Rightarrow \tau$ for types σ_i, τ; if $n = 0$ we just write τ. Type declarations are not types, but are used to "type" function symbols. All types can be expressed in the form $\sigma_1 \Rightarrow \ldots \Rightarrow \sigma_n \Rightarrow \iota$ (with $n \geq 0$ and $\iota \in \mathcal{B}$). The *order* of a type is $order(\iota) = 0$ if $\iota \in \mathcal{B}$, and $order(\sigma \Rightarrow \tau) = \max(order(\sigma)+1, order(\tau))$. Extending this to type declarations, $order([\sigma_1 \times \ldots \times \sigma_n] \Rightarrow \tau) = \max(order(\sigma_1) + 1, \ldots, order(\sigma_n) + 1, order(\tau))$.

We assume a set $\mathcal{V}$ of infinitely many typed variables for each type, and a set $\mathcal{F}$ disjoint from $\mathcal{V}$ which consists of function symbols, each equipped with a type declaration. *Terms* over $\mathcal{F}$ are those expressions s for which we can infer $s : \sigma$ for some type σ using the clauses:

(var)	$x : \sigma$	if $x : \sigma \in \mathcal{V}$
(app)	$s \cdot t : \tau$	if $s : \sigma \Rightarrow \tau$ and $t : \sigma$
(abs)	$\lambda x.\, s : \sigma \Rightarrow \tau$	if $x : \sigma \in \mathcal{V}$ and $s : \tau$
(fun)	$f(s_1, \ldots, s_n) : \tau$	if $f : [\sigma_1 \times \ldots \times \sigma_n] \Rightarrow \tau \in \mathcal{F}$ and $s_1 : \sigma_1, \ldots, s_n : \sigma_n$

Note that a function symbol $f : [\sigma_1 \times \ldots \times \sigma_n] \Rightarrow \tau$ takes exactly n arguments, and τ is not necessarily a base type (a type declaration gives the *arity* of the symbol). λ binds occurrences of variables as in the λ-calculus. Terms are considered modulo α-conversion; bound variables are renamed if necessary. Variables which are not bound are called *free*, and the set of free variables of s is denoted $FV(s)$. Application is left-associative, so $s \cdot t \cdot u$ should be read $(s \cdot t) \cdot u$. Terms constructed without clause (fun) are also called *(simply-typed) λ-terms*.

A *substitution* $[\vec{x} := \vec{s}]$, with $\vec{x}$ and $\vec{s}$ finite vectors of equal length, is the homomorphic extension of the type-preserving mapping $\vec{x} \mapsto \vec{s}$ from variables to terms. Substitutions are denoted γ, δ, and the result of applying γ to a term s is denoted $s\gamma$. The *domain* $\mathsf{dom}(\gamma)$ of $\gamma = [\vec{x} := \vec{s}]$ is $\{\vec{x}\}$. Substituting does not bind free variables. A *context* $C[]$ is a term with a single occurrence of a special symbol $\square_\sigma$. The result of replacing $\square_\sigma$ in $C[]$ by a term s of type σ is denoted $C[s]$. Free variables may be captured; if $C[] = \lambda x. \square_\sigma$ then $C[x] = \lambda x. x$.

Rules and Rewriting. A *rewrite rule* is a pair of terms $l \to r$ such that l and r have the same type and all free variables of r also occur in l. In [19] some termination-preserving transformations on the general format of AFS-rules are presented; using these results, we may additionally assume that l has the form $f(l_1, \dots, l_n) \cdot l_{n+1} \cdots l_m$ (with $f \in \mathcal{F}$ and $m \geq n \geq 0$), that l has no subterms $x \cdot s$ with x a free variable, and that neither l nor r have a subterm $(\lambda x. s) \cdot t$. Given a set of rules $\mathcal{R}$, the *rewrite* or *reduction relation* $\to_\mathcal{R}$ on terms is given by the following clauses:

`(rule)`	$C[l\gamma]$	$\to_\mathcal{R}$	$C[r\gamma]$	with $l \to r \in \mathcal{R}$, C a context, γ a substitution
(β)	$C[(\lambda x. s) \cdot t]$	$\to_\mathcal{R}$	$C[s[x := t]]$	with s, t terms, C a context

An *algebraic functional system (AFS)* is the combination of a set of terms and a rewrite relation on this set, and is usually specified by a pair $(\mathcal{F}, \mathcal{R})$, or just by a set $\mathcal{R}$ of rules. An AFS is terminating if there is no infinite reduction $s_1 \to_\mathcal{R} s_2 \to_\mathcal{R} \dots$

An AFS is *second-order* if the type declarations of all function symbols have order ≤ 2. In a second-order system, all free variables in the rules have order ≤ 1 (this follows by the restrictions on the left-hand side), and all bound variables have base type (this holds because free variables have order ≤ 1 and we have assumed that the rules do not contain β-redexes).

▶ **Example 2.1.** One of the examples considered in this paper is the AFS $\mathsf{shuffle}$. This (second-order) system for list manipulation has five function symbols, $\mathsf{nil} : \mathsf{natlist}$, $\mathsf{cons} : [\mathsf{nat} \times \mathsf{natlist}] \Rightarrow \mathsf{natlist}$, $\mathsf{append} : [\mathsf{natlist} \times \mathsf{natlist}] \Rightarrow \mathsf{natlist}$, $\mathsf{reverse} : [\mathsf{natlist}] \Rightarrow \mathsf{natlist}$, $\mathsf{shuffle} : [(\mathsf{nat} \Rightarrow \mathsf{nat}) \times \mathsf{natlist}] \Rightarrow \mathsf{natlist}$, and the following rules:

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{append}(\mathsf{cons}(h,t),l) & \to & \mathsf{cons}(h, \mathsf{append}(t,l)) & \mathsf{append}(\mathsf{nil},l) & \to & l \\
\mathsf{reverse}(\mathsf{cons}(h,t)) & \to & \mathsf{append}(\mathsf{reverse}(t), \mathsf{cons}(h,\mathsf{nil})) & \mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{shuffle}(F, \mathsf{cons}(h,t)) & \to & \mathsf{cons}(F \cdot h, \mathsf{shuffle}(F, \mathsf{reverse}(t))) & \mathsf{shuffle}(F, \mathsf{nil}) & \to & \mathsf{nil}
\end{array}$$

2.2 Reduction Pairs

To prove termination, modern approaches typically use *reduction pairs*, in one of three setups:

For *rule removal*, we consider a *strong reduction pair*: a pair $(\succsim, \succ)$ of a quasi-ordering and a well-founded ordering on terms, such that $\succsim$ and $\succ$ are *compatible*: $\succsim \cdot \succ$ is included in $\succ$ or $\succ \cdot \succsim$ is, both $\succsim$ and $\succ$ are *monotonic*, both $\succsim$ and $\succ$ are *stable* (preserved under substitution), and in the higher-order case, $\succsim$ *contains* β: $(\lambda x. s) \cdot t \succsim s[x := t]$.

If $\mathcal{R} = \mathcal{R}_1 \uplus \mathcal{R}_2$ and $l \succ r$ for rules in $\mathcal{R}_1$, and $l \succsim r$ for rules in $\mathcal{R}_2$, then there is no $\to_\mathcal{R}$-sequence which uses the rules in $\mathcal{R}_1$ infinitely often; this would contradict well-foundedness of $\succ$. Thus, $\to_\mathcal{R}$ is terminating if $\to_{\mathcal{R}_2}$ is terminating. In practice, we try to orient all rules with either $\succ$ or $\succsim$, and then remove those ordered with $\succ$ and continue with the rest.

The second setup, *dependency pairs*, is more sophisticated. In this approach, dependency pair chains are considered, which use infinitely many "dependency pair" steps at the top of a term. It is enough to orient the resulting constraints with a *weak reduction pair*: a pair $(\succsim, \succ)$ of a quasi-ordering and a compatible well-founded ordering where both are stable, and $\succsim$ is monotonic and contains β. The dependency pair approach was defined for first-order TRSs in [1], and has seen many extensions and improvements since. For higher-order rewriting, two variations exist: *static dependency pairs* [23] and *dynamic dependency pairs* [28, 21].

The static dependency pair approach is restricted to *plain function passing* systems; slightly simplified, whenever a higher-order variable F occurs in the right-hand side of a rule $f(l_1, \ldots, l_n) \to r$, then F is one of the l_i. Static dependency pairs may have variables in the right-hand side which do not occur in the left (such as a dependency pair $\mathsf{I}^\sharp(\mathsf{s}(n)) \to \mathsf{I}^\sharp(m)$), but always have the form $f^\sharp(l_1, \ldots, l_n) \to g^\sharp(r_1, \ldots, r_m)$. The static approach gives constraints of the form $l \succsim r$ or $l \succ r$ for dependency pairs $l \to r$, and $l \succsim r$ for rules $l \to r$.

The dynamic dependency pair approach is unrestricted, but right-hand sides of dependency pairs may be headed by a variable, e.g. $\mathsf{collapse}^\sharp(\mathsf{cons}(F, t)) \to F \cdot \mathsf{collapse}(t)$, and sometimes subterm steps are needed. Thus, the dynamic approach not only gives constraints $l \succ r$ or $l \succsim r$ for dependency pairs and $l \succsim r$ for rules, but also two further groups of constraints:

- $f(s_1, \ldots, s_n) \cdot t_1 \cdots t_m \succsim s_i \cdot \mathsf{c}_{\sigma_1} \cdots \mathsf{c}_{\sigma_{k_i}}$ if both sides have base type, $s_i : \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_{k_i} \Rightarrow \iota$ and f is a symbol in some fixed set S (the c_{σ_j} are special symbols which may occur in the right-hand sides of dependency pairs but do not occur in the rules)
- $s \cdot t_1 \cdots t_n \succsim t_i \cdot \mathsf{c}_{\sigma_1} \cdots \mathsf{c}_{\sigma_{k_i}}$ if both sides have base type, and $t_i : \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_{k_i} \Rightarrow \iota$

A third setup, which also uses a sort of reduction pair rather than the traditional reduction ordering, are the *monotonic semantic path orderings* from Borralleras and Rubio [5]. This method is based on a recursive path ordering, but uses a well-founded order on terms rather than a precedence on function symbols; this gives constraints of the form $s \succeq_I t$, $s \succeq_Q t$, $s \succ_Q t$, where $\succeq_I$ and $\succeq_Q$ are quasi-orderings and $s \succeq_I t$ implies $f(\ldots, s, \ldots) \succeq_Q f(\ldots, t, \ldots)$.

In this paper, we focus on the first two setups, which have been implemented in WANDA. However, the technique could be used with the monotonic semantic path ordering as well.

2.3 First-order Monotonic Algebras - Idea Sketch

In the first-order definition of monotonic algebras [9], terms are mapped to elements of a well-founded target domain $(A, >, \geq)$. This is done by choosing an interpretation function $\mathcal{J}(f)$ for all function symbols f that is monotonic w.r.t. $>$ and $\geq$, and extending this homomorphically to an interpretation $[\![\cdot]\!]$ of terms; for *polynomial interpretations*, $\mathcal{J}(f)$ is always a polynomial. If $[\![l]\!]_{\mathcal{J},\alpha} > [\![r]\!]_{\mathcal{J},\alpha}$ for all valuations α of the free variables of l, then $[\![C[l\gamma]]\!]_{\mathcal{J}} > [\![C[r\gamma]]\!]_{\mathcal{J}}$ for all contexts C and substitutions γ. Thus, the pair $(\succsim, \succ)$ where $s \succsim t$ if $[\![s]\!]_{\mathcal{J},\alpha} \geq [\![t]\!]_{\mathcal{J},\alpha}$ and $s \succ t$ if $[\![s]\!]_{\mathcal{J},\alpha} > [\![t]\!]_{\mathcal{J},\alpha}$ can be used as a strong reduction pair.

For example, to prove termination of the TRS consisting of the two append rules from Example 2.1, we might assign the following interpretation to the function symbols: $\mathcal{J}(\mathsf{nil}) = 2$, $\mathcal{J}(\mathsf{cons}) = \boldsymbol{\lambda} nm.n + m + 1$ and $\mathcal{J}(\mathsf{append}) = \boldsymbol{\lambda} nm.2 \cdot n + m + 1$. Here, the $\boldsymbol{\lambda}$ syntax indicates function creation: cons, for instance, is mapped to a function which takes two arguments, and returns their sum plus one. Calculating all $[\![l]\!]_{\mathcal{J},\alpha}, [\![r]\!]_{\mathcal{J},\alpha}$, and noting that $(\mathbb{N}, >, \geq)$ is a well-founded set and that all interpretations are monotonic functions, we see that the TRS is terminating because for all h, t, l: $4{+}l{+}1 > l$ (for the rule $\mathsf{append}(\mathsf{nil}, l) \to l$), and $2{\cdot}h{+}2{\cdot}t{+}2{+}l{+}1 > h{+}2{\cdot}t{+}l{+}1$ (for $\mathsf{append}(\mathsf{cons}(h, t), l) \to \mathsf{cons}(h, \mathsf{append}(t, l))$).

2.4 Weakly Monotonic Functionals

In higher-order rewriting we have to deal with infinitely many types (due to the type constructor $\Rightarrow$), a complication not present in first-order rewriting. As a consequence, it is not practical to map all terms to the same target set. A more natural interpretation would be, for instance, to map a functional term $\lambda x.\, s : \mathsf{o} \Rightarrow \mathsf{o}$ to an element of the function space $\mathbb{N} \Rightarrow \mathbb{N}$. However, this choice has problems of its own, since it forces the termination prover to deal with functions that absolutely nothing is known about. Instead, the target domain for interpreting terms, as proposed by van de Pol in [27], is the class of *weakly monotonic functionals.* To each type σ we assign a set $\mathcal{WM}_\sigma$ and two relations: a well-founded ordering $\sqsupset_\sigma$ and a quasi-ordering $\sqsupseteq_\sigma$. Intuitively, the elements of $\mathcal{WM}_{\sigma\Rightarrow\tau}$ are functions which preserve $\sqsupseteq$.

▶ **Definition 2.2** (Weakly Monotonic Functionals). [27, Def. 4.1.1] We assume given a *well-founded set*: a triple $\mathcal{A} = (A, >, \geq)$ of a non-empty set, a well-founded partial ordering on that set and a compatible quasi-ordering.[1] To each type σ we associate a set $\mathcal{WM}_\sigma$ of *weakly monotonic functionals of type* σ and two relations $\sqsupset_\sigma$ and $\sqsupseteq_\sigma$, defined inductively as follows:

For a base type ι, we have $\mathcal{WM}_\iota = A$; $\sqsupset_\iota \,=\, >$, and $\sqsupseteq_\iota \,=\, \geq$.

For a functional type $\sigma \Rightarrow \tau$, $\mathcal{WM}_{\sigma\Rightarrow\tau}$ consists of the functions f from $\mathcal{WM}_\sigma$ to $\mathcal{WM}_\tau$ such that: if $x \sqsupseteq_\sigma y$ then $f(x) \sqsupseteq_\tau f(y)$. Let $f \sqsupset_{\sigma\Rightarrow\tau} g$ iff $f(x) \sqsupset_\tau g(x)$, and $f \sqsupseteq_{\sigma\Rightarrow\tau} g$ iff $f(x) \sqsupseteq_\tau g(x)$ for all $x, y \in \mathcal{WM}_\sigma$.

Thus, $\mathcal{WM}_{\sigma\Rightarrow\tau}$ is a subset of the function space $\mathcal{WM}_\sigma \Rightarrow \mathcal{WM}_\tau$, consisting of functions which preserve $\sqsupseteq$. Note that both $\mathcal{WM}_\sigma$ and the relations $\sqsupset_\sigma$ and $\sqsupseteq_\sigma$ should be considered as parametrised with $\mathcal{A}$; the complete notation would be $(\mathcal{WM}^{\mathcal{A}}_\sigma, \sqsupset^{\mathcal{A}}_\sigma, \sqsupseteq^{\mathcal{A}}_\sigma)$. For readability, $\mathcal{A}$ will normally be omitted, as will the type denotations for the various $\sqsupset_\sigma$ and $\sqsupseteq_\sigma$ relations. The phrase "f is weakly monotonic" means that $f \in \mathcal{WM}_\sigma$ for some σ.

It is not hard to see that an element $\boldsymbol{\lambda} x_1 \dots x_n.P(x_1, \dots, x_n)$ of the function space $\mathcal{WM}_{\sigma_1} \Rightarrow \dots \Rightarrow \mathcal{WM}_{\sigma_n} \Rightarrow A$ is weakly monotonic if and only if:

$$\begin{gathered}\forall N_1, M_1 \in \mathcal{WM}_{\sigma_1}, \dots, N_n, M_n \in \mathcal{WM}_{\sigma_n} : \\ \text{if each } N_i \sqsupseteq M_i \text{ then } P(N_1, \dots, N_n) \sqsupseteq P(M_1, \dots, M_n)\end{gathered}$$

By Lemmas 4.1.3 and 4.1.4 in [27] we obtain several pleasant properties of $\sqsupseteq$ and $\sqsupset$:

▶ **Lemma 2.3.** *For all types σ, the relations $\sqsupset_\sigma$ and $\sqsupseteq_\sigma$ are compatible, $\sqsupset_\sigma$ is well founded, $\sqsupseteq_\sigma$ is reflexive, and both $\sqsupset_\sigma$ and $\sqsupseteq_\sigma$ are transitive.*

Comment: the definition in [27] actually assigns a different set $\mathcal{A}_\iota$ to each base type ι (although there must be an addition operator $+_{\iota,\kappa,\iota}$ for every pair of base types). We use the same set for all base types, as this gives a simpler definition, and it is not obvious that using different sets gives a stronger technique; we could for instance choose $\mathcal{A} = \mathcal{A}_\iota \uplus \mathcal{A}_\kappa$ instead.

Also, in [27] $\mathcal{WM}_{\sigma\Rightarrow\tau}$ consists of functions f in a larger function space $\mathcal{I}_\sigma \Rightarrow \mathcal{I}_\tau$[2] such that $f(x) \in \mathcal{WM}_\tau$ if $x \in \mathcal{WM}_\sigma$ and f preserves $\sqsupseteq$. Our definition is essentially equivalent; every function in $\mathcal{WM}_\sigma \Rightarrow \mathcal{WM}_\tau$ can be extended to a function in $\mathcal{I}_\sigma \Rightarrow \mathcal{I}_\tau$.

▶ **Example 2.4** (Some Examples of Weakly Monotonic Functionals).

1. *Constant Function:* For all $n \in A$ and types $\tau = \tau_1 \Rightarrow \dots \Rightarrow \tau_k \Rightarrow \iota$, let $n_\tau := \boldsymbol{\lambda}\vec{x}.n$. Then $n_\tau \in \mathcal{WM}_\tau$, since $n_\tau(N_1, \dots, N_k) = n \sqsupseteq n = n_\tau(M_1, \dots, M_k)$ if all $N_i \sqsupseteq M_i$.
2. *Lowest Value Function:* Suppose A has a minimal element 0 for the ordering $>$. Then for any type $\tau = \tau_1 \Rightarrow \dots \Rightarrow \tau_k \Rightarrow \iota$ the function $\boldsymbol{\lambda} f.f(\vec{0})$, which maps $f \in \mathcal{WM}_\tau$ to $f(0_{\tau_1}, \dots, 0_{\tau_k})$ (where each 0_{τ_i} is a constant function), is in $\mathcal{WM}_{\tau\Rightarrow\mathsf{o}}$ by induction on k.
3. *Maximum Function:* In the natural numbers, the function max which assigns to any two numbers the highest of the two is weakly monotonic, since $\max(a, b) \geq \max(a', b')$ if $a \geq a'$ and $b \geq b'$. For any type $\tau = \tau_1 \Rightarrow \dots \Rightarrow \tau_k \Rightarrow \iota$ (with $\iota \in \mathcal{B}$) let $\max_\tau(f, m) = \boldsymbol{\lambda} x_1 \dots x_k. \max(f(x_1, \dots, x_k), m)$. This function is in $\mathcal{WM}_{\tau\Rightarrow\iota\Rightarrow\tau}$ by induction on k.

[1] Van de Pol defines $\geq$ as the reflexive closure of $>$. In contrast, here we generalise the notion of a well-founded set to include an explicitly given compatible quasi-ordering $\geq$.

[2] Here, $\mathcal{I}_\iota = \mathcal{A}_\iota$ if $\iota \in \mathcal{B}$, and $\mathcal{I}_{\sigma\Rightarrow\tau}$ is the full function space $\mathcal{I}_\sigma \Rightarrow \mathcal{I}_\tau$.

The constant and lowest value function appear in [27]; the maximum function appears in [20].

▶ **Definition 2.5** (Interpreting a λ-Term to a Weakly Monotonic Functional)**.** Given a well-founded set $\mathcal{A} = (A, >, \geq)$, a simply-typed λ-term s and a *valuation* α which assigns to all variables $x : \sigma$ in $FV(s)$ an element of $\mathcal{WM}_\sigma$, let $[s]_\alpha$ be defined by the following clauses:

$$\begin{array}{lcll} [x]_\alpha & = & \alpha(x) & \text{if } x \in \mathcal{V} \\ [s \cdot t]_\alpha & = & [s]_\alpha([t]_\alpha) & \\ [\lambda x.\, s]_\alpha & = & \boldsymbol{\lambda} n.[s]_{\alpha \cup \{x \mapsto n\}} & \text{if } x \notin \mathsf{dom}(\alpha) \quad \text{(always applicable with } \alpha\text{-conversion)} \end{array}$$

Definition 2.5 is an instance of a definition in [27] which suffices for the extension to AFSs. By Lemma 3.2.1 and Proposition 4.1.5(1) in [27], we have:

▶ **Lemma 2.6** (Facts on λ-Term Interpretations)**.**

1. (Substitution Lemma) *Given a substitution $\gamma = [x_1 := s_1, \dots, x_n := s_n]$ and a valuation α whose domain does not include the x_i: $[s\gamma]_\alpha = [s]_{\alpha \circ \gamma}$. Here, $\alpha \circ \gamma$ is the valuation $\alpha \cup \{x_1 \mapsto [s_1]_\alpha, \dots, x_n \mapsto [s_n]_\alpha\}$.*

2. *If $s : \sigma$ is a simply-typed λ-term, then $[s]_\alpha \in \mathcal{WM}_\sigma$ for all valuations α.*

3 (Weakly and Extended) Monotonic Algebras for AFSs

The theory in [27] was defined for Nipkow's formalism of *Higher-order Rewrite Systems (HRSs)* [25], which differs in several ways from our *Algebraic Functional Systems.* Most importantly, in the setting of HRSs terms are equivalence classes modulo β; thus, the definitions in [27] are designed so that $[\![s]\!] = [\![t]\!]$ if s and t are equal modulo β. This is not convenient for AFSs, since then for instance $[\![(\lambda x.\, \mathsf{0}) \cdot t]\!] = [\![(\lambda x.\, \mathsf{0}) \cdot u]\!]$ regardless of t and u.

Fortunately, we do not need to redesign the whole theory for use with AFSs; rather, we can transpose the result using a transformation. We will need no more than Lemma 2.6.

Note: some of the results of this section have also been stated in [20], but the results there are limited to what is needed for the dynamic dependency pair approach; here, we are more general, by not fixing the interpretation of application and also studying strong monotonicity.

▶ **Definition 3.1** (Weakly Monotonic Algebras for AFSs)**.** A *weakly monotonic algebra* for an AFS with function symbols $\mathcal{F}$ consists of a well-founded set $\mathcal{A} = (A, >, \geq)$ and an *interpretation function* $\mathcal{J}$ which assigns an element of $\mathcal{WM}_{\sigma_1 \Rightarrow \ldots \Rightarrow \sigma_n \Rightarrow \tau}$ to all $f : [\sigma_1 \times \ldots \times \sigma_n] \Rightarrow \tau \in \mathcal{F}$, and a value in $\mathcal{WM}_{\sigma \Rightarrow \sigma}$ to the fresh symbol $@^\sigma$ for all functional types σ.

Given an algebra $(\mathcal{A}, \mathcal{J})$, a term s over $\mathcal{F}$ and a *valuation* α which assigns to all variables $x : \sigma$ in $FV(s)$ an element of $\mathcal{WM}_\sigma$, let $[\![s]\!]_{\mathcal{J},\alpha}$ be defined recursively as follows:

$$\begin{array}{lcll} [\![x]\!]_{\mathcal{J},\alpha} & = & \alpha(x) & \text{if } x \in \mathcal{V} \\ [\![f(s_1, \dots, s_n)]\!]_{\mathcal{J},\alpha} & = & \mathcal{J}(f)([\![s_1]\!]_{\mathcal{J},\alpha}, \dots, [\![s_n]\!]_{\mathcal{J},\alpha}) & \text{if } f \in \mathcal{F} \\ [\![s \cdot t]\!]_{\mathcal{J},\alpha} & = & \mathcal{J}(@^\sigma)([\![s]\!]_{\mathcal{J},\alpha}, [\![t]\!]_{\mathcal{J},\alpha}) & \text{if } s : \sigma \\ [\![\lambda x.\, s]\!]_{\mathcal{J},\alpha} & = & \boldsymbol{\lambda} n.[\![s]\!]_{\mathcal{J},\alpha \cup \{x \mapsto n\}} & \text{if } x \notin \mathsf{dom}(\alpha) \end{array}$$

This definition, which roughly follows the ideas of [27] and extends the definition of a weakly monotone algebra in [9] to the setting of AFSs, assigns to every function symbol and variable a weakly monotonic functional, and calculates the value of the term accordingly. For the purposes of the interpretation, application is treated as a function symbol $@^\sigma$. As in [27], the interpretation function $\mathcal{J}$ is separate from the valuation α, as we will quantify over α.

▶ **Example 3.2.** Consider the $\mathsf{shuffle}$ signature from Example 2.1, extended with symbols $\mathsf{0}$ and s for the natural numbers. Let $\mathcal{A} = (\mathbb{N}, >, \geq)$. By way of example, choose: $\mathcal{J}(\mathsf{0}) = 1$, $\mathcal{J}(\mathsf{s}) = \boldsymbol{\lambda} n.n + 2$, $\mathcal{J}(\mathsf{cons}) = \boldsymbol{\lambda} nm.n + m$, $\mathcal{J}(\mathsf{shuffle}) = \boldsymbol{\lambda} Fn.F(n)$ and $\alpha(z) = 37$. Then $[\![\mathsf{shuffle}(\lambda x.\, \mathsf{s}(x), \mathsf{cons}(\mathsf{s}(\mathsf{0}), z))]\!]_{\mathcal{J},\alpha} = [\![F(n)]\!]_{\mathcal{J},\{F \mapsto \boldsymbol{\lambda} m.m+2, n \mapsto 40\}} = 42$.

▶ **Lemma 3.3** (Weakly Monotonic Algebras for AFSs). *Let $(\mathcal{A}, \mathcal{J})$ be a weakly monotonic algebra for $\mathcal{F}$, and s, t terms over $\mathcal{F}$. For all valuations α as described in Definition 3.1:*

1. $[\![s]\!]_{\mathcal{J},\alpha} \in \mathcal{WM}_\sigma$ *if* $s : \sigma$.
2. $[\![s]\!]_{\mathcal{J},\alpha\circ\gamma} = [\![s\gamma]\!]_{\mathcal{J},\alpha}$ *(where* $\alpha \circ \gamma = \alpha \cup \{x \mapsto [\![\gamma(x)]\!]_{\mathcal{J},\alpha} \mid x \in \mathsf{dom}(\gamma)\}$*)*
3. *If* $[\![s]\!]_{\mathcal{J},\delta} \sqsupseteq [\![t]\!]_{\mathcal{J},\delta}$ *for all valuations* δ*, then* $[\![s\gamma]\!]_{\mathcal{J},\alpha} \sqsupseteq [\![t\gamma]\!]_{\mathcal{J},\alpha}$.
   If $[\![s]\!]_{\mathcal{J},\delta} \sqsupset [\![t]\!]_{\mathcal{J},\delta}$ *for all valuations* δ*, then* $[\![s\gamma]\!]_{\mathcal{J},\alpha} \sqsupset [\![t\gamma]\!]_{\mathcal{J},\alpha}$.
4. *If* $[\![s]\!]_{\mathcal{J},\delta} \sqsupseteq [\![t]\!]_{\mathcal{J},\delta}$ *for all valuations* δ*, then* $[\![C[s]]\!]_{\mathcal{J},\alpha} \sqsupseteq [\![C[t]]\!]_{\mathcal{J},\alpha}$.

Proof. The proof proceeds by translating (arbitrary) terms to simply-typed λ-terms, and then reusing the original result. Interpretation of function symbols ($\mathcal{J}$) is translated to assignment of variables (α), and application is treated as a function symbol.

Consider the following transformation:

$$\begin{array}{rclcrcll} \varphi(x) & = & x \;\; (x \in \mathcal{V}) & & \varphi(f(s_1,\dots,s_n)) & = & x_f \cdot \varphi(s_1) \cdots \varphi(s_n) & (f \in \mathcal{F}) \\ \varphi(\lambda x.\, s) & = & \lambda x.\, \varphi(s) & & \varphi(s \cdot t) & = & x_{@,\sigma} \cdot \varphi(s) \cdot \varphi(t) & (s : \sigma) \end{array}$$

Here, the x_f is a new variable of type $\sigma_1 \Rightarrow \dots \Rightarrow \sigma_n \Rightarrow \tau$ for $f : [\sigma_1 \times \dots \times \sigma_n] \Rightarrow \tau \in \mathcal{F}$, and $x_{@,\sigma}$ is a variable of type $\sigma \Rightarrow \sigma$. For any substitution γ, let γ^φ denote the substitution $[x := \varphi(\gamma(x)) \mid x \in \mathsf{dom}(\gamma)]$ (the x_f are left alone). We make the following observations:

(**) $\varphi(s\gamma) = \varphi(s)\gamma^\varphi$ *for all substitutions* γ.

(***) $[\![s]\!]_{\mathcal{J},\alpha} = [\varphi(s)]_\delta$*, if* $\delta(x) = \alpha(x)$ *for* $x \in FV(s)$*,* $\delta(x_f) = \mathcal{J}(f)$*,* $\delta(x_{@,\sigma}) = \mathcal{J}(@^\sigma)$

Both statements hold by a straightforward induction on the form of s.

(1) holds by (***) and Lemma 2.6(2). **(2)** holds because $[\![s\gamma]\!]_{\mathcal{J},\alpha} = [\varphi(s\gamma)]_\delta$ by (***), $= [\varphi(s)\gamma^\varphi]_\delta$ by (**), $= [\varphi(s)]_{\delta\circ\gamma^\varphi}$ by Lemma 2.6(1), which is exactly $[\![s]\!]_{\mathcal{J},\alpha\circ\gamma}$ by (***). **(3)** holds by (2): $[\![s\gamma]\!]_{\mathcal{J},\alpha} = [\![s]\!]_{\mathcal{J},\alpha\circ\gamma}$ by (2), $\sqsupseteq [\![t]\!]_{\mathcal{J},\alpha\circ\gamma} = [\![t\gamma]\!]_{\mathcal{J},\alpha}$, and similar for $\sqsupset$. **(4)** holds by a straightforward induction on the form of C (this result has no counterpart in [27]). ◀

The theory so far allows us to use weakly monotonic algebras in a *weak reduction pair*.

▶ **Theorem 3.4.** *Let a weakly monotonic algebra* $(\mathcal{A}, \mathcal{J})$ *be given such that always* $\mathcal{J}(@^\sigma) \sqsupseteq \boldsymbol{\lambda} fn.f(n)$*, and define the pair* $(\succsim, \succ)$ *by:* $s \succsim t$ *if* $[\![s]\!]_{\mathcal{J},\alpha} \sqsupseteq [\![t]\!]_{\mathcal{J},\alpha}$ *for all valuations* α*, and* $s \succ t$ *if* $[\![s]\!]_{\mathcal{J},\alpha} \sqsupset [\![t]\!]_{\mathcal{J},\alpha}$ *for all* α*. Then* $(\succsim, \succ)$ *is a weak reduction pair.*

Proof. $(\succsim, \succ)$ is a compatible combination of a quasi-ordering and a well-founded ordering by Lemma 2.3 and $\succsim$ is monotonic by Lemma 3.3(4). Also, $\succsim$ contains `beta`: for all valuations α, $[\![(\lambda x.\, s) \cdot t]\!]_{\mathcal{J},\alpha} = \mathcal{J}(@^\sigma)([\![\lambda x.\, s]\!]_{\mathcal{J},\alpha}, [\![t]\!]_{\mathcal{J},\alpha}) \sqsupseteq [\![\lambda x.\, s]\!]_{\mathcal{J},\alpha}([\![t]\!]_{\mathcal{J},\alpha})$ by assumption, which equals $[\![s]\!]_{\mathcal{J},\alpha\cup\{x\mapsto[\![t]\!]_{\mathcal{J},\alpha}\}} = [\![s]\!]_{\mathcal{J},\alpha\circ[x:=t]}$, and this equals $[\![s[x := t]]\!]_{\mathcal{J},\alpha}$ by Lemma 3.3(2). ◀

Comment: if we choose $\mathcal{J}(@^\sigma) = \boldsymbol{\lambda} fn.f(n)$, we have a system very similar to the one used for simply-typed λ-calculus (and HRSs). By not fixing the interpretation of $@^\sigma$ we have a choice, which, depending on the setting (rule removal, static dependency pairs, dynamic dependency pairs) may be essential; we will see different choices in Examples 3.5, 3.6 and 4.4.

▶ **Example 3.5.** Using the static dependency pair framework of [23] to deal with $\mathsf{shuffle}$, we obtain several sets of requirements. HORPO [17] runs into trouble with the dependency pair $\mathsf{shuffle}^\sharp(F, \mathsf{cons}(h,t)) \to \mathsf{shuffle}^\sharp(F, \mathsf{reverse}(t))$, where we need a weak reduction pair satisfying:

$$\begin{array}{rcllrcl} \mathsf{shuffle}^\sharp(F, \mathsf{cons}(h,t)) & \succ & \mathsf{shuffle}^\sharp(F, \mathsf{reverse}(t)) & & & & \\ \mathsf{append}(\mathsf{cons}(h,t), l) & \succsim & \mathsf{cons}(h, \mathsf{append}(t, l)) & & \mathsf{append}(\mathsf{nil}, l) & \succsim & l \\ \mathsf{reverse}(\mathsf{cons}(h,t)) & \succsim & \mathsf{append}(\mathsf{reverse}(t), \mathsf{cons}(h, \mathsf{nil})) & & \mathsf{reverse}(\mathsf{nil}) & \succsim & \mathsf{nil} \end{array}$$

Using Theorem 3.4, we choose the following interpretation $\mathcal{J}$ in the natural numbers:

$$\begin{array}{lllllllll} \mathcal{J}(\mathsf{shuffle}^\sharp) & = & \boldsymbol{\lambda} fn.n & \mathcal{J}(\mathsf{cons}) & = & \boldsymbol{\lambda} nm.m+1 & \mathcal{J}(\mathsf{nil}) & = & 0 \\ \mathcal{J}(\mathsf{reverse}) & = & \boldsymbol{\lambda} n.n & \mathcal{J}(\mathsf{append}) & = & \boldsymbol{\lambda} nm.n+m & \mathcal{J}(@^\sigma) & = & \boldsymbol{\lambda} fn.f(n) \text{ for all } \sigma \end{array}$$

Quantifying over the valuation, it suffices to show that for all $F \in \mathcal{WM}_{\mathsf{nat}\Rightarrow\mathsf{nat}}, h, t \in \mathbb{N}$: $t+1 > t,\ t+l+1 \geq t+l+1,\ t+1 \geq t+1,\ l \geq l,\ 0 \geq 0$. This is obviously the case!

▶ **Example 3.6.** For a case where we cannot choose $\mathcal{J}(@^\sigma) = \boldsymbol{\lambda} fn.f(n)$, consider collapse:

$$\begin{array}{rclrclrcl} 0 & : & \mathsf{nat} & \mathsf{min} & : & [\mathsf{nat} \times \mathsf{nat}] \Rightarrow \mathsf{nat} & \mathsf{cons} & : & [(\mathsf{nat} \Rightarrow \mathsf{nat}) \times \mathsf{flist}] \Rightarrow \mathsf{flist} \\ \mathsf{s} & : & [\mathsf{nat}] \Rightarrow \mathsf{nat} & \mathsf{diff} & : & [\mathsf{nat} \times \mathsf{nat}] \Rightarrow \mathsf{nat} & \mathsf{build} & : & [\mathsf{nat}] \Rightarrow \mathsf{flist} \\ \mathsf{nil} & : & \mathsf{flist} & \mathsf{gcd} & : & [\mathsf{nat} \times \mathsf{nat}] \Rightarrow \mathsf{nat} & \mathsf{collapse} & : & [\mathsf{flist}] \Rightarrow \mathsf{nat} \end{array}$$

$$\begin{array}{rclrcl} \mathsf{min}(x,0) & \to & 0 & \mathsf{gcd}(\mathsf{s}(x),0) & \to & \mathsf{s}(x) \\ \mathsf{min}(0,x) & \to & 0 & \mathsf{gcd}(0,\mathsf{s}(x)) & \to & \mathsf{s}(x) \\ \mathsf{min}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{min}(x,y)) & \mathsf{gcd}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{gcd}(\mathsf{diff}(x,y),\mathsf{s}(\mathsf{min}(x,y))) \\ \mathsf{diff}(x,0) & \to & x & \mathsf{build}(0) & \to & \mathsf{nil} \\ \mathsf{diff}(0,x) & \to & x & \mathsf{build}(\mathsf{s}(x)) & \to & \mathsf{cons}(\lambda y.\,\mathsf{gcd}(y,x),\mathsf{build}(x)) \\ \mathsf{diff}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{diff}(x,y) & \mathsf{collapse}(\mathsf{nil}) & \to & 0 \\ & & & \mathsf{collapse}(\mathsf{cons}(F,t)) & \to & F \cdot \mathsf{collapse}(t) \end{array}$$

This AFS is not plain function passing, so we cannot use static dependency pairs. Using dynamic dependency pairs, HORPO runs into trouble when faced with the constraints:

$$\begin{array}{lcl} \mathsf{collapse}^\sharp(\mathsf{cons}(F,t)) & \underset{(\sim)}{\succsim} & F \cdot \mathsf{collapse}(t) \\ \mathsf{collapse}^\sharp(\mathsf{cons}(F,t)) & \underset{(\sim)}{\succsim} & \mathsf{collapse}^\sharp(t) \qquad l \succsim r \quad \text{for all rules } l \to r \text{ listed above} \end{array}$$

The $\underset{(\sim)}{\succsim}$ relation denotes that the constraint can either be oriented with $\succsim$ or with $\succ$; to make progress, at least one of these constraints must be oriented with $\succ$. Recall that in the dynamic dependency pair approach the constraints must be satisfied with a reduction pair that also has $s \cdot t_1 \cdots t_n \succsim t_i \cdot c_1 \cdots c_m$ if both sides have base type, for fresh constants c_j; moreover, we must have $\mathsf{gcd}(x,y) \succsim x, y$. To guarantee this, we choose $\mathcal{J}(@^{\sigma\Rightarrow\tau}) = \boldsymbol{\lambda} fn.\max_\tau(f(n), n(\vec{0}))$, where $n(\vec{0})$ and $\max_\tau$ were defined in Example 2.4. Then $\mathcal{J}(@^{\sigma\Rightarrow\tau}) \sqsupseteq \boldsymbol{\lambda} fn.f(n)$, and if we assign $\mathcal{J}(c_j) = 0_\sigma$ for $c_j : \sigma$, then $[\![s \cdot \vec{t}]\!]_{\mathcal{J},\alpha} \sqsupseteq [\![t_i \cdot \vec{c}]\!]_{\mathcal{J},\alpha}$ is indeed satisfied. Additionally, let $\mathcal{J}(0) = \mathcal{J}(\mathsf{nil}) = 0,\ \mathcal{J}(\mathsf{diff}) = \mathcal{J}(\mathsf{gcd}) = \boldsymbol{\lambda} nm.n+m,\ \mathcal{J}(\mathsf{s}) = \mathcal{J}(\mathsf{build}) = \boldsymbol{\lambda} n.3 \cdot n,\ \mathcal{J}(\mathsf{min}) = \boldsymbol{\lambda} nm.0,\ \mathcal{J}(\mathsf{collapse}) = \boldsymbol{\lambda} n.n,\ \mathcal{J}(\mathsf{collapse}^\sharp) = \boldsymbol{\lambda} n.n+1$ and $\mathcal{J}(\mathsf{cons}) = \boldsymbol{\lambda} fn.f(n)+n$.

With this interpretation, we have $l \succsim r$ for all rules. Moreover, $[\![\mathsf{collapse}^\sharp(\mathsf{cons}(F,t))]\!]_{\mathcal{J},\alpha} = 1+F(t)+t > \max(F(t),t) = [\![F \cdot \mathsf{collapse}(t)]\!]_{\mathcal{J},\alpha}$ and $[\![\mathsf{collapse}^\sharp(\mathsf{cons}(F,t))]\!]_{\mathcal{J},\alpha} = 1+F(t)+t \geq 1+t = [\![\mathsf{collapse}^\sharp(t)]\!]_{\mathcal{J},\alpha}$. As required, we can remove one dependency pair (the first one).

Strong Monotonicity. To use weakly monotonic algebras in the setting of rule removal, we shall need an additional requirement: $\sqsupset$ must be monotonic. This is achieved by posing a restriction on $\mathcal{J}$: each $\mathcal{J}(f)$ should be *strongly monotonic*:

▶ **Definition 3.7** (Strongly Monotonic Functional). An element f of $\mathcal{WM}_{\sigma_1\Rightarrow\ldots\Rightarrow\sigma_n\Rightarrow\iota}$ is *strongly monotonic in argument i* if for all $N_1 \in \mathcal{WM}_{\sigma_1}, \ldots, N_n \in \mathcal{WM}_{\sigma_n}$ and $M_i \in \mathcal{WM}_{\sigma_i}$ we have: $f(N_1, \ldots, N_i, \ldots, N_n) \sqsupset f(N_1, \ldots, M_i, \ldots, N_n)$ if $N_i \sqsupset M_i$.

For first- and second-order functions, strong monotonicity corresponds with the notion *strict* in [27]. For higher-order functions, the definition of [27] is more permissive. We have chosen to use strong monotonicity because the strictness requirement significantly complicates the theory of [27], and most common examples of higher-order systems are second-order. Strongly monotonic functionals exist for all types, e.g. $\boldsymbol{\lambda} x_1 \dots x_n . x_1(\vec{0}) + \dots + x_n(\vec{0}) \in \mathcal{WM}_{\tau_1 \Rightarrow \dots \Rightarrow \tau_n \Rightarrow \iota}$.

An *extended monotonic algebra* is a weakly monotonic algebra where each $\mathcal{J}(@^\sigma)$ is strongly monotonic in its first two arguments,[3] and for $f : [\sigma_1 \times \dots \times \sigma_n] \Rightarrow \tau \in \mathcal{F}$ also $\mathcal{J}(f)$ is strongly monotonic in its first n arguments. This notion extends the corresponding definition from [9] for the first-order setting to the setting of AFSs. We obtain:

▶ **Theorem 3.8.** *Let an extended monotonic algebra $(\mathcal{A}, \mathcal{J})$ be given such that always $\mathcal{J}(@^\sigma) \sqsupseteq \boldsymbol{\lambda} fn.f(n)$; the pair $(\succsim, \succ)$ from Theorem 3.4 is a strong reduction pair.*

Proof. It is a weak reduction pair by Theorem 3.4, and strongly monotonic because $[\![C[s]]\!]_{\mathcal{J},\alpha} \sqsupset [\![C[t]]\!]_{\mathcal{J},\alpha}$ for all α whenever $[\![s]\!]_{\mathcal{J},\alpha} \sqsupset [\![t]\!]_{\mathcal{J},\alpha}$ for all α (an easy induction). ◀

4 Higher-Order Polynomial Interpretations

It remains to be seen how to *find* suitable polynomial interpretations, preferably automatically. In this section, we will discuss the class of *higher-order polynomials over* $\mathbb{N}$, a specific subclass of the weakly monotonic functionals with $(\mathbb{N}, >, \geq)$ as a well-founded base set. In the following, we will see how suitable polynomials can be found automatically.

▶ **Definition 4.1** (Higher-Order Polynomial over $\mathbb{N}$)**.** For a set $X = \{x_1 : \sigma_1, \dots, x_n : \sigma_n\}$ of variables, each equipped with a type, the set $Pol(X)$ of *higher-order polynomials* in X is given by the following clauses:

- if $n \in \mathbb{N}$, then $n \in Pol(X)$;
- if $p_1, p_2 \in Pol(X)$, then $p_1 + p_2 \in Pol(X)$ and $p_1 \cdot p_2 \in Pol(X)$;
- if $x_i : \tau_1 \Rightarrow \dots \Rightarrow \tau_m \Rightarrow \iota \in X$ with $\iota \in \mathcal{B}$, and $p_1 \in Pol^{\tau_1}(X), \dots, p_m \in Pol^{\tau_m}(X)$, then $x_i(p_1, \dots, p_m) \in Pol(X)$;
  - here, $Pol^\iota(X) = Pol(X)$ for base types ι, and $Pol^{\sigma \Rightarrow \tau}(X)$ contains functions $\boldsymbol{\lambda} y.p \in \mathcal{WM}_\sigma$ with $p \in Pol^\tau(X \cup \{y\})$.

We do not fix the set X. A *higher-order polynomial* is an element of any $Pol(X)$.

Noting that $\mathcal{WM}_\sigma = \mathcal{WM}_\tau$ if σ and τ have the same "form" (so are equal modulo renaming of base types), the following lemma holds for all $\iota \in \mathcal{B}$:

▶ **Lemma 4.2.** *If $p \in Pol(\{x_1 : \sigma_1, \dots, x_n : \sigma_n\})$, then $\boldsymbol{\lambda} x_1 \dots x_n . p \in \mathcal{WM}_{\sigma_1 \Rightarrow \dots \Rightarrow \sigma_n \Rightarrow \iota}$.*

Proof. It is easy to see that $+$ and $\cdot$ are weakly monotonic. Taking this into account, the lemma follows quickly with induction on the size of p, using Lemma 2.6(2). For the variable case, if $\boldsymbol{\lambda} \vec{y}.p_i \in Pol^{\tau_i}(\{\vec{x}\})$, then $p_i \in Pol(\{\vec{x}, \vec{y}\})$, so the induction hypothesis applies. ◀

Higher-order polynomials are typically represented in the form $a_1 + \dots + a_n$ (with $n \geq 0$), where each a_i is a *higher-order monomial*: an expression of the form $b \cdot c_1 \cdots c_m$, where $b \in \mathbb{N}$ and each c_i is either a base-type variable x or a function application $x(\boldsymbol{\lambda} \vec{y_1}.p_1, \dots, \boldsymbol{\lambda} \vec{y_k}.p_k)$ with all p_j higher-order polynomials again. Examples of higher-order polynomials over the natural numbers are for instance 0 and $3 + 5 \cdot x^2 \cdot y + F(37 + x)$. To find a *strongly monotonic* functional, it suffices to include, for all variables, a monomial containing only that variable:

[3] Note that e.g. $\mathcal{J}(@^{o \Rightarrow o \Rightarrow o})$ is an element of the function space $\mathcal{WM}_{o \Rightarrow o \Rightarrow o} \Rightarrow \mathcal{WM}_o \Rightarrow \mathcal{WM}_o \Rightarrow \mathcal{WM}_o$; a function which takes *three* arguments. It need not be strongly monotonic in its 3rd argument, because we think of application as a symbol $@^{\sigma \Rightarrow \tau} : [(\sigma \Rightarrow \tau) \times \sigma] \Rightarrow \tau$ of arity *2*, where τ may be functional.

▶ **Lemma 4.3.** *Let $P(x_1,\dots,x_n)$ be a higher-order polynomial of the form $p_1(\vec{x})+\dots+p_m(\vec{x})$, where all $p_i(\vec{x})$ are higher-order monomials. Then $\boldsymbol{\lambda}\vec{x}.P(\vec{x})$ is strongly monotonic in argument i if there is some p_j of the form $a \cdot x_i(\vec{b}(\vec{x}))$, where $a \in \mathbb{N}^+$.*

Proof. Let $x_i \sqsupset x_i'$, so also $x_i \sqsupseteq x_i'$ (since $> \subseteq \geq$). Let $\vec{x} := x_1,\dots,x_i,\dots,x_l$ and $\vec{x'} := x_1,\dots,x_i',\dots,x_l$. All p_k are weakly monotonic by Lemma 4.2, so $p_k(\vec{x}) \sqsupseteq p_k(\vec{x'})$. Since $p_j(\vec{x}) \sqsupset p_j(\vec{x'})$ and $+$ is strongly monotonic, indeed $P(\vec{x}) \sqsupset P(\vec{x'})$. ◀

▶ **Example 4.4.** For rule removal on the AFS shuffle from Ex. 2.1, consider the interpretation:

$$\begin{array}{rclrcl}
\mathcal{J}(\mathsf{append}) & = & \boldsymbol{\lambda}nm.n+m & \mathcal{J}(\mathsf{cons}) & = & \boldsymbol{\lambda}nm.n+m+3 \\
\mathcal{J}(\mathsf{reverse}) & = & \boldsymbol{\lambda}n.n+1 & \mathcal{J}(\mathsf{nil}) & = & 0 \\
\mathcal{J}(\mathsf{shuffle}) & = & \boldsymbol{\lambda}Fn.2n+F(0)+nF(n)+1 & \mathcal{J}(@^\sigma) & = & \boldsymbol{\lambda}fn\vec{m}.f(n,\vec{m})+n(\vec{0})\ (**)
\end{array}$$

(**) Here, $n(\vec{0})$ is the "lowest value" function from Ex. 2.4. With this interpretation, which is a strongly monotonic polynomial interpretation by Lemma 4.3, all rules are oriented with $\succsim$, and the two shuffle rules and the reverse(nil) one even with $\succ$. Only for the main shuffle rule this is non-trivial to see; here we have the constraint: $F(h+t+3)+tF(h+t+3)+F(h+t+3)+[h+hF(h+t+3)+3F(h+t+3)]+2 > F(h)+tF(t+1)+F(t+1)$. This holds by weak monotonicity of F: since $h+t+3 \geq h$ always holds, we must have $F(h+t+3) \geq F(h)$ as well, and similarly we see that $tF(h+t+3) \geq tF(t+1)$ and $F(h+t+3) \geq F(t+1)$.

5 Automation

To demonstrate that the approach is automatable, we have made a proof-of-concept implementation of polynomial interpretations in the higher-order termination tool WANDA. The implementation only tries simple parametric shapes, does not use heuristics, and is limited to second-order AFSs – a limitation which excludes but 5 out of the 156 higher-order benchmarks in the current *termination problem database (TPDB)*,[4] as the class of second-order systems is very common.[5] Even with this minimal implementation, the combination of polynomial interpretations with dependency pairs can handle about 75% of the TPDB.

To find polynomial interpretations automatically, WANDA uses the following steps:

1. assign every function symbol a higher-order polynomial with *parameters* as coefficients;
2. for all requirements $l \underset{(\sim)}{\succsim} r$ and $l \succsim r$, calculate $[\![l]\!]_{\mathcal{J},\alpha}$ and $[\![r]\!]_{\mathcal{J},\alpha}$ as a function on parameters and variables – this gives constraints $P_i \underset{(\,)}{\geq} Q_i$ and $P_i \geq Q_i$;
3. introduce a parameter o_i for all constraints of the form $P_i \underset{(\,)}{\geq} Q_i$, and replace these constraints by $P_i \geq Q_i + o_i$; if we also introduce the constraints $o_1+\dots+o_n \geq 1$ then, when all constraints are satisfied, at least one $\underset{(\,)}{\geq}$ constraint is strictly oriented;
4. simplify the constraints until they no longer contain variables;
5. impose maximum values on the search space of the parameters and use a non-linear constraint solver to find a solution for the constraints.

These steps are detailed below, with an AFS rule for the function map as a running example.

[4] See `http://termination-portal.org/wiki/TPDB` for details on this standard database.
[5] The restriction to second-order systems is not essential, but it makes the code easier in a number of places: we can avoid representing function-polynomials $\boldsymbol{\lambda}\vec{x}.P(\vec{x})$, stick to simple interpretation shapes, and we do not have max in the left-hand side of constraints. Mostly, the restriction is present because of the low number of available benchmarks of order 3 or higher, which makes it hard to select suitable interpretation shapes, and not initially worth the added implementation effort.

5.1 Choosing Parametric Polynomial Interpretations

The module for polynomial interpretations in WANDA is called in three contexts: rule removal, the dynamic dependency pair framework and the static dependency pair framework. In the first case, function interpretations must be strongly monotonic, in the second case they have to satisfy a subterm property, and in the third there are no further restrictions.

To start, every function symbol $f : [\sigma_1 \times \ldots \times \sigma_n] \Rightarrow \sigma_{n+1} \Rightarrow \ldots \Rightarrow \sigma_m \Rightarrow \iota \in \mathcal{F}$ is assigned a function of the form $\boldsymbol{\lambda} x_1 \ldots x_m . p_1 + p_2 + a$, where a is a parameter and:

- p_1 has the form $a_1 \cdot x_1(0, \ldots, 0) + \ldots + a_m \cdot x_m(0, \ldots, 0)$, where the a_i are parameters (this is well-typed because we work in a second-order system);
  - in the rule removal setting, we add requirements: $a_1 \geq 1, \ldots, a_n \geq 1$;
  - in the dynamic dependency pairs setting, we add requirements: $a_{n+1} \geq 1, \ldots, a_m \geq 1$.
- $p_2 = q_1 + \ldots + q_k$, where each q_j has the form $c_j \cdot x_{i_1} \cdots x_{i_k} \cdot x_j(x_{i_1}, \ldots, x_{i_k}) + d_j \cdot x_j(x_{i_1}, \ldots, x_{i_k})$, with c_j, d_j parameters, the x_{i_l} first-order variables, and x_j a higher-order variable; every combination of a higher-order variable with first-order variables occurs.[6]

We must also choose an interpretation of $@^\sigma$ for all types. Rather than using a parametric interpretation, we observe that application occurs mostly on the right-hand side of constraints. There, we often have (sub-)terms $F \cdot s_1 \cdots s_n$ with F a free variable; on the left-hand side, such subterms do not occur, nor can we have applications headed by an abstraction or bound variable (in a second-order system, bound variables have base type). Only applications of the form $f(s_1, \ldots, s_n) \cdot s_{n+1} \cdots s_m$ occur on the left; since function symbols usually have a base type as output type, this is a rare situation. Thus, we fix the interpretation of $@^\sigma$ for all types to be as small as possible. Note that we must have $\mathcal{J}(@^\sigma) \sqsupseteq \boldsymbol{\lambda} fn.f(n)$ by Theorem 3.4, and $\mathcal{J}(@^\sigma)$ may have to be strongly monotonic, or satisfy a subterm property.

- in the rule removal setting, $\mathcal{J}(@^\sigma) = \boldsymbol{\lambda} fn\vec{m}.f(n, \vec{m}) + n(\vec{0})$;
- in the dynamic dependency pairs setting, $\mathcal{J}(@^\sigma) = \boldsymbol{\lambda} fn\vec{m}. \max(f(n, \vec{m}), n(\vec{0}))$;
- in the static dependency pairs setting, $\mathcal{J}(@^\sigma) = \boldsymbol{\lambda} fn\vec{m}.f(n, \vec{m})$.

In the rule removal setting this choice together with the constraints on the parameters guarantees that all $\mathcal{J}(f)$ are strongly monotonic in the arguments required by the definition of an extended monotonic algebra and Theorem 3.8. In the dynamic dependency pairs setting, we obtain the required subterm property as demonstrated in Example 3.6. Moreover, in this setting always $[\![f(s_1, \ldots, s_n) \cdot s_{n+1} \cdots s_m]\!]_{\mathcal{J},\alpha} = \max(\mathcal{J}(f)([\![s_1]\!]_{\mathcal{J},\alpha}, \ldots, [\![s_n]\!]_{\mathcal{J},\alpha}, [\![s_{n+1}]\!]_{\mathcal{J},\alpha}, \ldots, [\![s_m]\!]_{\mathcal{J},\alpha}), [\![s_{n+1}]\!]_{\mathcal{J},\alpha}(\vec{0}), \ldots, [\![s_m]\!]_{\mathcal{J},\alpha}(\vec{0})) = \mathcal{J}(f)([\![s_1]\!]_{\mathcal{J},\alpha}, \ldots, [\![s_m]\!]_{\mathcal{J},\alpha})$ by the restriction on the parameters. Thus, although we now also need to deal with the max-operator, it will only ever occur on the right-hand side of a constraint! Since this avoids the need for conditional constraints as used in [11] (without losing any power), it both simplifies the automation and creates smaller constraints.

From these parametric higher-order polynomials, we calculate the interpretations of terms, and simplify the resulting higher-order polynomials into a sum of monomials. For the constraints $l \underset{(\sim)}{\succ} r$, in general we use constraints $[\![l]\!]_{\mathcal{J},\alpha} \geq [\![r]\!]_{\mathcal{J},\alpha} + o$ for some fresh *bit* o (a parameter whose value ranges over $\{0, 1\}$), and require that the sum of these bits is positive.

▶ **Example 5.1** (Running Example). To demonstrate the technique, consider rule removal on the recursive rule of the common map example, which gives the constraint $\mathsf{map}(F, \mathsf{cons}(h, t)) \underset{(\sim)}{\succ}$

[6] In case the constraint solver does not find a solution for this interpretation shape, WANDA additionally includes non-linear monomials $c_{i,j} \cdot x_i \cdot x_j$ (where $i < j$) without functional variables in the parametric higher-order polynomials and tries again. In general, here one can use arbitrary parametric polynomials.

$\mathsf{cons}(F \cdot h, \mathsf{map}(F,t))$. We assign: $\mathcal{J}(\mathsf{cons}) = \boldsymbol{\lambda} nm.a_1 \cdot n + a_2 \cdot m + a_3$ and $\mathcal{J}(\mathsf{map}) = \boldsymbol{\lambda} fn.a_4 \cdot f(0) + a_5 \cdot n + a_6 \cdot n \cdot f(n) + a_7$.[7] This leads to the following constraints:

- $a_1, a_2, a_4, a_5 \geq 1$, $o_1 \geq 1$ (we could also immediately replace o_1 by 1).
- $a_7 + a_3 \cdot a_5 + a_1 \cdot a_5 \cdot h + a_2 \cdot a_5 \cdot t + a_4 \cdot F(0) + a_1 \cdot a_6 \cdot h \cdot F(a_1 \cdot h + a_2 \cdot t + a_3) + a_2 \cdot a_6 \cdot t \cdot F(a_1 \cdot h + a_2 \cdot t + a_3) + a_3 \cdot a_6 \cdot F(a_1 \cdot h + a_2 \cdot t + a_3) \geq a_3 + a_2 \cdot a_7 + o_1 + a_1 \cdot h + a_2 \cdot a_5 \cdot t + a_2 \cdot a_4 \cdot F(0) + a_1 \cdot F(h) + a_6 \cdot t \cdot F(t)$

5.2 Simplifying Polynomial Requirements

We obtain requirements that contain variables as well as parameters; they should be read as "there exist a_i, o_k such that for all h, t, F the inequalities hold". To avoid dealing with claims over all possible numbers or functions we simplify the requirements until they contain no more variables. To a large extent, these simplifications correspond to the ones used with automations of polynomial interpretations for first-order rewriting [6], but higher-order variables in function application present an extra difficulty. To deal with application of higher-order variables, we will use Lemma 5.2:

▶ **Lemma 5.2.** *Let F be a weakly monotonic functional and all p, q, p_i, q_i, s_i, r_i polynomials.*

1. $F(r_1, \ldots, r_k) \cdot p \geq F(s_1, \ldots, s_k) \cdot q$ *if* $r_1 \geq s_1, \ldots, r_k \geq s_k, p \geq q$.

2. $r_1 \cdot p_1 + \ldots + r_n \cdot p_n \geq s_1 \cdot q_1 + \ldots + s_m \cdot q_m$ *if there are* $e_{i,j}$ *for* $1 \leq i \leq n, 1 \leq j \leq m$ *with:*

   a. *for all i:* $r_i \geq e_{i,1} + \ldots + e_{i,m}$*;*

   b. *for all j:* $e_{1,j} + \ldots + e_{n,j} \geq s_j$*;*

   c. *either* $e_{i,j} = 0$ *or* $p_i \geq q_j$.

Proof. **(1)** holds by weak monotonicity of F. As for **(2)**, $r_1 \cdot p_1 + \ldots + r_n \cdot p_n \geq \sum_{i=1}^n \sum_{j=1}^m e_{i,j} \cdot p_i$ by **(a)**, and since $e_{i,j} = 0$ whenever not $p_i \geq q_j$ by **(c)**, $\sum_{i=1}^n \sum_{j=1}^m e_{i,j} \cdot p_i \geq \sum_{i=1}^n \sum_{j=1}^m e_{i,j} \cdot q_j = \sum_{j=1}^m \sum_{i=1}^n e_{i,j} \cdot q_j$. Using **(b)**, $\sum_{j=1}^m \sum_{i=1}^n e_{i,j} \cdot q_j \geq \sum_{j=1}^m s_j \cdot q_j$ as required. ◀

Lemma 5.2, together with some observations used in the first-order case, supplies the theory we need to simplify the requirements to constraints which do not contain any variables. Here, a "component" of a monomial $a_1 \cdots a_n$ is any of the a_i (but p is not a component of $F(p)$).

1. Do standard simplifications on the constraints, for instance replacing $3 \cdot F(n) \geq F(n) + a_1 \cdot n$ by $2 \cdot F(n) \geq a_1 \cdot n$ and $p + B \cdot p$ by $(B+1) \cdot p$ if B is a *known* constant, and removing monomials $0 \cdot p$. Remove constraints $p \geq 0$ and $p \geq p$ which always hold.
2. Split constraints $0 \geq p_1 + \ldots + p_n$ into the n constraints $0 \geq p_i$. Remove constraints $0 \geq a$ where a is a single parameter, and replace a by 0 everywhere else.
   This is valid because, in the natural numbers, $0 \geq a$ implies $a = 0$, and $0 + \ldots + 0 = 0$.
3. Replace constraints $P \geq Q[\max(r,s)]$ by the two constraints $P \geq Q[r]$ and $P \geq Q[s]$.
   This is valid because for any valuation $Q[\max(r,s)]$ equals $Q[r]$ or $Q[s]$.
4. Given a constraint $p_1 + \ldots + p_n \geq p_{n+1} + \ldots + p_m$ where some, but not all, of the monomials p_i contain a component x or $x(\vec{q})$ for some fixed variable x, let A contain the indices i of those monomials p_i which have x or $x(\vec{q})$. Replace the constraint by the two constraints $\sum_{i \in A, i \leq n} p_i \geq \sum_{i \in A, i > n} p_i$ and $\sum_{i \notin A, i \leq n} p_i \geq \sum_{i \notin A, i > n} p_i$. For example, splitting on n, the constraint $3 \cdot n \cdot m + a_2 \cdot F(a_3 \cdot n + a_4) \geq 2 + a_7 \cdot m + F(n)$ is split into $3 \cdot n \cdot m \geq 0$ and $a_2 \cdot F(a_3 \cdot n + a_4) \geq 2 + a_7 \cdot m + F(n)$; subsequently, splitting on F, the latter is split into $a_2 \cdot F(a_3 \cdot n + a_4) \geq F(n)$ and $0 \geq 2 + a_7 \cdot m$.
   This is valid because $p_1 + p_2 \geq q_1 + q_2$ certainly holds if $p_1 \geq q_1$ and $p_2 \geq q_2$.

[7] To ease presentation, in contrast to WANDA here we do not use an addend $a_i \cdot f(n)$ for map.

5. If all non-zero monomials on either side of a constraint have a component x, "divide out" x. For example, replace the constraint $a_1 \cdot n + n \cdot n \cdot f(a_3, n) \geq n + a_3 \cdot n$ by $a_1 + n \cdot f(a_3, n) \geq 1 + a_3$, and replace $0 \geq a_5 \cdot m$ by $0 \geq a_5$.
This is valid because $p \cdot n \geq q \cdot n$ holds if $p \geq q$ (cf. the absolute positiveness criterion [15]).
6. Replace a constraint $s \cdot x_1(p_{1,1}, \ldots, p_{1,k_1}) \cdots x_n(p_{n,1}, \ldots, p_{n,k_n}) \geq s \cdot x_1(q_{1,1}, \ldots, q_{1,k_1}) \cdots x_n(q_{n,1}, \ldots, q_{n,k_n})$ by the constraints $s \cdot p_{i,j} \geq s \cdot q_{i,j}$ for all i, j.
This is valid by Lemma 5.2(1) and case analysis whether $s = 0$ or not.
7. Let $p_1, \ldots, p_n, q_1, \ldots, q_m$ be monomials of the form $x_1(\vec{r_1}), \ldots, x_k(\vec{r_k})$, for fixed $x_1, \ldots, x_k$. Replace a constraint $r_1 \cdot p_1 + \ldots + r_n \cdot p_n \geq s_1 \cdot q_1 + \ldots + s_m \cdot q_m$ with $n, m \geq 1$ by the following constraints, where the $e_{i,j}$ are fresh parameters:
   for $1 \leq i \leq n$: $r_i \geq e_{i,1} + \ldots + e_{i,m}$
   for $1 \leq j \leq m$: $e_{1,j} + \ldots + e_{n,j} \geq s_j$
   for $1 \leq i \leq n,\ 1 \leq j \leq m$: $e_{i,j} \cdot p_i \geq e_{i,j} \cdot q_j$ (which can be handled with clause 6)
This is valid by Lemma 5.2(2).[8]

It is easy to see that while a constraint still has variables in it, we can apply clauses to simplify or split it (taking into account that max does not appear in the left-hand side of a constraint), and that the clauses also terminate on a system without variables. These simplifications are not complete: for example, a universally valid constraint $F(n) \cdot n \geq F(1) \cdot n$ is split into constraints $n \geq n$ (which holds), and $n \geq 1$ (which does not).

► **Example 5.3.** Let us simplify the constraints from Example 5.1. First, using clause 4 to group monomials by their variables, we obtain:
- $a_1, a_2, a_4, a_5, o_1 \geq 1$
- $a_7 + a_3 \cdot a_5 \geq a_3 + a_2 \cdot a_7 + o_1$
- $a_1 \cdot a_5 \cdot h \geq a_1 \cdot h$
- $a_2 \cdot a_5 \cdot t \geq a_2 \cdot a_5 \cdot t$
- $a_4 \cdot F(0) + a_3 \cdot a_6 \cdot F(a_1 \cdot h + a_2 \cdot t + a_3) \geq a_2 \cdot a_4 \cdot F(0) + a_1 \cdot F(h)$
- $a_1 \cdot a_6 \cdot h \cdot F(a_1 \cdot h + a_2 \cdot t + a_3) \geq 0$
- $a_2 \cdot a_6 \cdot t \cdot F(a_1 \cdot h + a_2 \cdot t + a_3) \geq a_6 \cdot t \cdot F(t)$

The 4th and 6th requirements are trivial and can be removed with clause 1. After dividing away the non-functional variables using clause 5 we have the following constraints left:
- $a_1, a_2, a_4, a_5, o_1 \geq 1$
- $a_7 + a_3 \cdot a_5 \geq a_3 + a_2 \cdot a_7 + o_1$
- $a_1 \cdot a_5 \geq a_1$
- $a_4 \cdot F(0) + a_3 \cdot a_6 \cdot F(a_1 \cdot h + a_2 \cdot t + a_3) \geq a_2 \cdot a_4 \cdot F(0) + a_1 \cdot F(h)$
- $a_2 \cdot a_6 \cdot F(a_1 \cdot h + a_2 \cdot t + a_3) \geq a_6 \cdot F(t)$

The first three are completely simplified. Clauses 7 and 6 replace the last two constraints by:
- $a_4 \geq e_{1,1} + e_{1,2}, \quad a_3 \cdot a_6 \geq e_{2,1} + e_{2,2}, \quad e_{1,1} + e_{2,1} \geq a_2 \cdot a_4, \quad e_{1,2} + e_{2,2} \geq a_1$
- $e_{1,1} \cdot 0 \geq e_{1,1} \cdot 0, \quad e_{1,2} \cdot 0 \geq e_{1,2} \cdot h$
- $e_{2,1} \cdot (a_1 \cdot h + a_2 \cdot t + a_3) \geq e_{2,1} \cdot 0, \quad e_{2,2} \cdot (a_1 \cdot h + a_2 \cdot t + a_3) \geq e_{2,2} \cdot h$
- $a_2 \cdot a_6 \geq k_{1,1}, \quad k_{1,1} \geq a_6, \quad k_{1,1} \cdot (a_1 \cdot h + a_2 \cdot t + a_3) \geq k_{1,1} \cdot t$

Using clauses 1, 4 and 5, we can simplify the constraints further, and obtain:

[8] In the cases where $n = 1$ or $m = 1$, some of these parameters are unnecessary; for instance, if $n = 1$, we can safely fix $e_{1,j} = s_j$ for all j. Our actual implementation uses a few of such special-case optimisations.

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
a_1, a_2, a_4, a_5, o_1 & \geq & 1 & a_1 \cdot a_5 & \geq & a_1 & a_2 \cdot a_6 & \geq & k_{1,1} \\
a_7 + a_3 \cdot a_5 & \geq & a_3 + a_2 \cdot a_7 + o_1 & a_4 & \geq & e_{1,1} & k_{1,1} & \geq & a_6 \\
a_3 \cdot a_6 & \geq & e_{2,1} + e_{2,2} & e_{2,2} & \geq & a_1 & k_{1,1} \cdot a_2 & \geq & k_{1,1} \\
e_{1,1} + e_{2,1} & \geq & a_2 \cdot a_4 & e_{2,2} \cdot a_1 & \geq & e_{2,2} & & &
\end{array}$$

Thus, using a handful of clauses, the requirements are simplified to a number of constraints with parameters over the natural numbers. In the actual WANDA implementation a few small optimisations are used; for example, some simplifications are combined, and if $\max(r, s)$ occurs more than once in the same polynomial, all occurrences are replaced by r or s at the same time. However, these optimisations make no fundamental difference to the method.

After imposing bounds on the search space, we can solve the resulting non-linear constraints using standard SAT- [10] or SMT-based [3] techniques (WANDA uses a SAT encoding similar to [10] with the solver MiniSAT [7] as back-end). If the problem is satisfiable, the solver returns values for all parameters, so it is easy to see which requirements have been oriented with $>$. For map, the solver could provide for example the solution $a_7 = e_{2,1} = 0$, $a_1 = a_2 = a_3 = a_4 = a_6 = e_{1,1} = e_{2,2} = k_{1,1} = o_1 = 1$, and $a_5 = 2$. This results in the interpretation $\mathcal{J}(\mathsf{cons}) = \boldsymbol{\lambda} nm.n + m + 1$ and $\mathcal{J}(\mathsf{map}) = \boldsymbol{\lambda} fn.f(0) + 2 \cdot n + n \cdot f(n)$.

6 Experiments

For an empirical evaluation of our contributions, we conducted a number of experiments with our implementation in WANDA using an Intel Xeon 5140 CPU with four cores at 2.33 GHz (cf. also `http://aprove.informatik.rwth-aachen.de/eval/HOPOLO/` for details on the evaluation). As underlying benchmark set, we used the 156 examples from the higher-order category of the TPDB version 8.0.1 together with Examples 2.1 and 3.6. WANDA invokes the SAT solver MiniSAT [7] and the first-order termination prover AProVE [14] as back-ends. As in the Termination Competition, we imposed a 60 second timeout per example.

The module for polynomial interpretation is called potentially twice with different polynomial shapes, as described at the start of Section 5.1 (cf. Footnote 6). The search space for the parameters is $\{0, \ldots, 3\}$. Our first experiment is designed to analyse the impact of polynomial interpretations coupled with a higher-order dependency pair framework.

Configuration	YES	NO	MAYBE	TIMEOUT	Avg. time
WANDA full	124	9	23	2	3.19 s
WANDA no poly	119	9	30	0	2.40 s
WANDA no horpo	118	9	28	3	3.59 s

Figure 1 Experimental results of full WANDA with and without polynomials or horpo.

Fig. 1 shows the results of *WANDA full*, which includes both polynomial interpretations and HORPO, the other main class of orderings implemented by WANDA (other than that, WANDA only uses the subterm criterion as an ordering-based technique). They are compared to versions of WANDA where either polynomials or HORPO are disabled. Although WANDA already scored highest in the Termination Competition of 2011, adding the contributions of this paper gives an additional 5 examples on the benchmark set. It is interesting to note that even without HORPO, WANDA with polynomials can still show termination of 118 examples.

Using the contributions of [12], WANDA delegates the first-order part of a higher-order rewrite system to the first-order termination tool AProVE, where it is commonplace to use polynomial interpretations. The setup of our second experiment deals with the impact of higher-order polynomial interpretations if WANDA does not use a first-order tool.

Configuration	YES	NO	MAYBE	TIMEOUT	Avg. time
WANDA no [12] full	118	9	29	2	2.32 s
WANDA no [12] no poly	107	9	42	0	1.09 s
WANDA no [12] no horpo	111	9	35	3	2.89 s

Figure 2 Experimental results of WANDA without first-order back-end.

Fig. 2 juxtaposes the results of WANDA without the first-order prover AProVE in three configurations. We see that if we disable the first-order back-end, the increase in power by polynomial interpretations goes up from 5 examples in the first experiment to 11 examples in the second. Thus, the gain of using a first-order tool can at least partially be compensated by using native higher-order polynomial interpretations.

Our third experiment investigates the impact of higher-order polynomial interpretations if no dependency pairs are used (which also excludes first-order termination tools). Here we compare to the version of HORPO implemented in WANDA.

Configuration	YES	NO	MAYBE	TIMEOUT	Avg. time
Rule Removal both	76	9	70	3	3.60 s
Rule Removal horpo	69	9	80	0	1.01 s
Rule Removal poly	47	9	97	5	3.64 s

Figure 3 Experimental results of WANDA with rule removal (and without dependency pairs).

Using just rule removal, HORPO clearly trumps polynomial interpretations. However, in part this may be due to the limited choice in interpretation shapes this first implementation of polynomial interpretations supports.

Discussion. Analysing the termination problem database, it is perhaps not surprising that the gain from using polynomial interpretations in the first experiment is not larger: the majority of the benchmarks which WANDA cannot already handle is non-terminating, or not known to be terminating (for example, state-of-the-art first-order tools cannot prove termination of the first-order part). For others, type-conscious methods such as *accessibility* (see e.g. [2]) are required; the method described in this paper ignores differences in base types. For cases where polynomial interpretations are needed, but only for the (truly) first-order part, passing this first-order part [12] to a modern first-order tool already suffices – as is evident by comparing the numbers in the first and second experiments. With higher-order polynomial interpretations, we have gained three out of the remaining seven benchmarks.

7 Conclusion

In this paper, we have extended the termination method of weakly monotonic algebras to the class of AFSs, simplifying definitions and adding the theory to use algebras with rule removal and dependency pairs; some efforts towards this were previously made in [20], but only for the setting of dynamic dependency pairs. Then, we introduced the class of higher-order polynomial interpretations, and discussed how suitable interpretations can be found automatically. The implementation of polynomial interpretations increases the power of WANDA by a respectable five benchmarks, including the two examples in this paper.

Thus, weakly monotonic algebras form an elegant method for proving termination by hand and, as demonstrated by the implementation in WANDA and the results of the experiments, a feasible automatable termination method as well.

Future Work. We have by no means reached the limit of what can be achieved with this technique: we might consider different interpretation shapes, possibly coupled with heuristics to determine a suitable shape. Or we may go beyond polynomials; we could for instance use max in function interpretations as done in e.g. [11], or (for a truly higher-order alternative), use repeated function application; this leads to interpretations like $\boldsymbol{\lambda} nmf. \max(m, f^n(m))$.[9]

Another alley to explore is to combine polynomial interpretations with type interpretations: rather than collapsing all base types into one, we might *translate* them, e.g. mapping a base type funclist to $\mathsf{o} \Rightarrow \mathsf{o}$. WANDA already does this in very specific cases, and one could simultaneously search for polynomial interpretations and for a type interpretation – this could parallel the search for type orderings in implementations of the recursive path ordering [2].

Moreover, in the first-order world, there are many more applications of monotonic algebras, e.g. matrix, arctic, rational, real and integer interpretations... There is no obvious reason why these methods cannot be lifted to the higher-order case as well!

References

1 T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.

2 F. Blanqui, J.-P. Jouannaud, and A. Rubio. The computability path ordering: The end of a quest. In *Proc. CSL 2008*, LNCS 5213, pages 1–14, 2008.

3 C. Borralleras, S. Lucas, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. SAT modulo linear arithmetic for solving polynomial constraints. *Journal of Automated Reasoning*, 48(1):107–131, 2012.

4 C. Borralleras and A. Rubio. THOR – a higher-order termination tool. `http://www.lsi.upc.edu/~albert/term.html`.

5 C. Borralleras and A. Rubio. A monotonic higher-order semantic path ordering. In *Proc. LPAR 2001*, LNAI 2250, pages 531–547, 2001.

6 E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325–363, 2005.

7 N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. SAT 2003*, LNCS 2919, pages 502–518, 2004.

8 J. Endrullis. Jambox. `http://joerg.endrullis.de/`.

9 J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning*, 40(2-3):195–220, 2008.

10 C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proc. SAT 2007*, LNCS 4501, pages 340–354, 2007.

11 C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. Maximal termination. In *Proc. RTA 2008*, LCNS 5117, pages 110–125, 2008.

12 C. Fuhs and C. Kop. Harnessing first order termination provers using higher order dependency pairs. In *Proc. FroCoS 2011*, LNAI 6989, pages 147–162, 2011.

13 C. Fuhs and C. Kop. Polynomial interpretations for higher-order rewriting. Technical Report `arXiv:1203.5754 [cs.LO]`, 2012. `http://arxiv.org/abs/1203.5754`.

14 J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. IJCAR 2006*, LNAI 4130, pages 281–286, 2006.

15 H. Hong and D. Jakuš. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21(1):23–38, 1998.

[9] The max is essential in this interpretation because $\boldsymbol{\lambda} nmf.f^n(m)$ is not weakly monotonic. A case analysis whether $m \geq f(m)$ or $f(m) \geq m$ shows that $\boldsymbol{\lambda} nmf. \max(m, f^n(m))$ *is* weakly monotonic.

16 J.-P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *Proc. LICS 1991*, pages 350–361, 1991.

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

18 C. Kop. WANDA – a higher order termination tool. `http://few.vu.nl/~kop/code.html`.

19 C. Kop. Simplifying algebraic functional systems. In *Proc. CAI 2011*, LNCS 6742, pages 201–215, 2011.

20 C. Kop and F. van Raamsdonk. Higher order dependency pairs for algebraic functional systems. In *Proc. RTA 2011*, LIPIcs 10, pages 203–218, 2011.

21 C. Kop and F. van Raamsdonk. Dynamic dependency pairs for algebraic functional systems. *Logical Methods in Computer Science*, 2012. Special Issue of the 22nd International Conference on Rewriting Techniques and Applications (RTA 2011). To appear.

22 M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. Tyrolean Termination Tool 2. In *Proc. RTA 2009*, LNCS 5595, pages 295–304, 2009.

23 K. Kusakari, Y. Isogai, M. Sakai, and F. Blanqui. Static dependency pair method based on strong computability for higher-order rewrite systems. *IEICE Transactions on Information and Systems*, 92(10):2007–2015, 2009.

24 D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.

25 T. Nipkow. Higher-order critical pairs. In *Proc. LICS 1991*, pages 342–349, 1991.

26 J. van de Pol. Termination proofs for higher-order rewrite systems. In *Proc. HOA 1993*, LNCS 816, pages 305–325, 1994.

27 J.C. van de Pol. *Termination of Higher-order Rewrite Systems*. PhD thesis, University of Utrecht, 1996.

28 M. Sakai, Y. Watanabe, and T. Sakabe. An extension of the dependency pair method for proving termination of higher-order rewrite systems. *IEICE Transactions on Information and Systems*, E84-D(8):1025–1032, 2001.

29 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

30 Wiki. Termination portal. `http://www.termination-portal.org/`.

On Soundness Conditions for Unraveling Deterministic Conditional Rewrite Systems*

Karl Gmeiner, Bernhard Gramlich, and Felix Schernhammer

Faculty of Informatics, TU Wien, Austria
`{gmeiner,gramlich,felixs}@logic.at`

Abstract

We study (un)soundness of transformations of conditional term rewriting systems (CTRSs) into unconditional term rewriting systems (TRSs). The focus here is on analyzing (un)soundness of so-called unravelings, the most basic and natural class of such transformations. We extend our previous analysis from normal 1-CTRSs to the more general class of deterministic CTRSs (DCTRSs) where extra variables in right-hand sides of rules are allowed to a certain extent. We prove that the previous soundness results based on weak left-linearity and on right-linearity can be extended from normal 1-CTRSs to DCTRSs. Counterexamples show that such an extension to DCTRSs does not work for the previous criteria which were based on confluence and on non-erasingness, not even for right-stable systems. Yet, we prove weaker versions of soundness criteria based on confluence and on non-erasingness. Finally, we compare our approach and results with other recently established soundness criteria for unraveling DCTRSs.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Conditional term rewriting system (CTRS), deterministic CTRS, transformation, simulation, soundness

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.193

Category Regular Research Paper

1 Introduction and Overview

1.1 Background and Motivation

Unconditional term rewriting systems (TRSs) are very well studied and enjoy many nice properties. However, often TRSs are insufficient to appropriately model computations or specifications, since the applicability of rules is inherently conditional. Thus, conditional term rewriting systems (CTRSs) naturally arise in many settings and examples. Since conditional rewriting is known to be much more involved than unconditional rewriting, both in theory and in practice, an attractive approach to analysis and implementation of CTRSs consists in transforming them into unconditional TRSs where ordinary (unconditional) rewriting can simulate the original conditional computations.

There exists abundant literature on conditional rewriting, cf. e.g. [16, 3] and also on transforming CTRSs into TRSs, where computation is sometimes restricted by further mechanism like membership constraints, context-sensitivity or imposed reduction strategies so as to avoid reductions which have no counterpart in the conditional setting (cf. e.g. [6, 8, 14, 13, 17, 19]). Typically, *completeness* of such transformations (w.r.t. reduction) is easily obtained

* The first author was supported by a grant of the *Vienna PhD School of Informatics*, the last author by a grant of the *Austrian Academy of Sciences* (ÖAW-DOC grant No. 22361).

and proved (by construction). However, *soundness* is much more difficult to analyze and to achieve. The reason simply is that in the encoding much more fine-grained rewrite computations are possible and potentially dangerous, since they may lead to reductions between terms in the original signature which have not been possible in the original CTRS.

Concerning soundness of transformations using restricted versions of unconditional rewriting various positive and negative results are known. But concerning soundness of transformation approaches using unrestricted unconditional rewriting little was known until recently, and mostly only for the simplest class of CTRSs, namely oriented normal 1-CTRSs, and only for the simplest class of such transformations, the *unravelings* ([10]).[1]

The first important soundness result for unravelings is due to Marchiori [10] who showed that unraveling left-linear normal 1-CTRSs is sound (cf. also [16]). In [12] and — based on this paper — recently in [15] Nishida et al. presented an analysis of (a slightly optimized form of) unraveling *deterministic* CTRSs (DCTRSs), an interesting subclass of 3-CTRSs, where they showed that soundness is guaranteed if the transformed system is either left-linear or both right-linear and non-erasing. In [8] we have shown that a few other sufficient soundness criteria for the case of normal 1-CTRSs exist, including confluence and non-erasingness. Moreover, we could show there that instead of left-linearity even *weak left-linearity* is sufficient for soundness. In weakly left-linear systems one may have non-left-linear rules like $eq(x, x) \to true$ where the non-linear variables are erased.

Here we extend our analysis for normal 1-CTRSs also to the practically important class of DCTRSs, and finally compare the approach and results with the ones of [15].

1.2 Contributions

First we discuss *simultaneous* versus *sequential* unravelings. A careful analysis reveals that instead of *simultaneously* unraveling normal 1-CTRSs as in [16, 8], a *sequential* unraveling is also perfectly possible thus enforcing a simulated evaluation of the conditions from left to right. In the case of DCTRSs sequentially unraveling still yields an ordinary unconditional TRS, although the original DCTRS may have extra variables in the right-hand sides and the conditions of rules. A careful analysis reveals that for normal 1-CTRSs all soundness results for the simultaneous case from [8] extend to the sequential case.

Our main results for transforming DCTRSs into TRSs via sequential unravelings are the following:

- We show that various tempting extensions of the soundness results for normal 1-CTRSs to DCTRSs do not hold, even for quite restricted sub-classes of DCTRSs, in particular potential criteria based on confluence and on non-erasingness (as in [8]), cf. Example 3.1).
- Our main positive result is that weak left-linearity of a DCTRS is already sufficient for soundness of its transformed version (Theorem 3.28).
- For non-erasingness, we show that we obtain soundness only for a restricted class of DCTRSs (Theorem 3.16).
- Furthermore, we show that right-linearity of the transformed system is also a sufficient condition for soundness (Theorem 3.11).
- Regarding confluence, we only get a weaker soundness criterion w.r.t. reduction to normal form (Theorem 3.11).

[1] The very idea of *unravelings* is actually much older and appears already e.g. in [4], though in a specialized form (for function definitions).

The rest of the paper is structured as follows. In Section 2 we present the necessary technical and conceptual background. The main Section 3 contains the soundness analysis. In Section 4 we discuss related work, especially [12, 15], and promising directions for future research.

Due to space restrictions, for some results we only sketch the proof or give its main idea and the underlying intuition. Full proofs of these results are provided in the long Technical Report version [9] of this paper.

2 Preliminaries

We assume familiarity with the basic concepts and notations of abstract reductions systems (ARSs) and (conditional) term rewriting systems (CTRSs) (cf. e.g. [2], [16], [3]).

2.1 Basics

For the sake of readability we recall some notions and notations: The set of (non-variable, variable) positions of a term s is denoted as $\mathcal{P}os(s)$ ($\mathcal{FP}os(s)$, $\mathcal{VP}os(s)$). $\mathsf{root}(s)$ denotes the root symbol of the term s. Throughout the paper $\mathcal{V}$ denotes a countably infinite set of variables. x, y, z denote variables from $\mathcal{V}$. By $\mathcal{V}ar(s)$ we denote the set of variables of a term s. If $X = \{x_1, \ldots, x_n\}$ is a set of variables, we denote by $\vec{X}$ the sequence of all variables in X in some arbitrary but fixed order (e.g. with: if $X \subseteq Y$, then $\vec{X}$ is a prefix of $\vec{Y}$). By $|s_1, \ldots, s_n|_x$ we mean the number of occurrences of the (variable) symbol x in $s_1, \ldots, s_n$.

A term rewriting system $\mathcal{R}$ is a pair $(\mathcal{F}, R)$ of a signature and a set of rewrite rules over this signature. Slightly abusing notation we also write $\mathcal{R}$ instead of R (leaving the signature implicit). A rewrite system $\mathcal{R}$ is called *non-erasing* (NE) if $\mathcal{V}ar(r) = \mathcal{V}ar(l)$ for all $l \to r \in \mathcal{R}$, and *left-linear* (LL) (*right-linear* (RL)) if every $x \in \mathcal{V}ar(l)$ ($x \in \mathcal{V}ar(r)$) occurs exactly once in l (r) for all rules $l \to r \in \mathcal{R}$. The function $\mathsf{lin} : \mathcal{T} \to \mathcal{T}$ renames non-linear variables into fresh new variables while keeping the linear ones.

We denote a rewrite step from a term u to a term v at position p in a rewrite system $\mathcal{R}$ with a rule α from $\mathcal{R}$ as $u \to_{p,\mathcal{R},\alpha} v$. We skip $p, \mathcal{R}$ or α if they are clear from the context or of no relevance. The parallel reduction at positions $P \subseteq \mathcal{P}os(u)$ is denoted as $u \mathrel{+\!\!\!\!\!\!\!\to}_{P,\mathcal{R}} v$.

The set of *one-step descendants* of a (subterm) position p of a term u w.r.t. a (one-step) reduction $u = C[s]_p \to_q v$ is the set of positions in v given by $\{p\}$, if $q \geq p$ or $q \parallel p$; $\{q.o'.p' \mid t|_q = l\sigma, l|_o \in \mathcal{V}ar(l), q.o.p' = p, l|_o = r|_{o'}\}$, if $q < p$ and (a superterm of) s is bound to a variable in the matching of $u|_q$ with the lhs of the applied rule; and $\emptyset$, otherwise. Slightly abusing terminology, when $u = C[s]_p \to_q v$ with set $\{p_1, \ldots, p_k\}$ of one-step descendants of p in v, we also say that $u|_p$ has the one-step descendants $v|_{p_i}$ in v. The *descendant relation* (w.r.t. given derivations) is obtained as the (reflexive-)transitive closure of the one-step descendant relation. The relation of (one-step) *ancestors* of a subterm position (w.r.t. a given reduction sequence) is the inverse relation of the (one-step) descendant relation.

A conditional term rewriting system $\mathcal{R}$ (over some signature $\mathcal{F}$) consists of rules $l \to r \Leftarrow c$ where $l \notin \mathcal{V}$ and c is a conjunction of equations $s_i = t_i$. Equality in the conditions may be interpreted (recursively) e.g. as $\leftrightarrow^*$ (semi-equational case), as $\downarrow$ (join case), or as $\to^*$ (oriented case). In the latter case, if all right-hand sides of conditions are ground terms that are irreducible w.r.t. the unconditional version $\mathcal{R}_u = \{l \to r \mid l \to r \Leftarrow c \in \mathcal{R}\}$ of $\mathcal{R}$, the system is said to be a *normal* CTRS. Subsequently, unless otherwise stated, we will always consider oriented CTRSs.

According to the distribution of variables, a conditional rule $l \to r \Leftarrow c$ may satisfy (1) $\mathcal{V}ar(r) \cup \mathcal{V}ar(c) \subseteq \mathcal{V}ar(l)$, (2) $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$, (3) $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l) \cup \mathcal{V}ar(c)$, or (4) no variable constraints at all. If all rules of a CTRS $\mathcal{R}$ are of type (i), $1 \leq i \leq 4$, we say that $\mathcal{R}$ is an i-CTRS. Given a conditional rewrite rule $l \to r \Leftarrow c$ and a variable x such that $x \in \mathcal{V}ar(r) \cup \mathcal{V}ar(c)$ but $x \notin \mathcal{V}ar(l)$, we say that x is an *extra variable.* An oriented 3-CTRS $\mathcal{R}$ is called *deterministic* if for every conditional rule $l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$ we have $\mathcal{V}ar(s_i) \subseteq \mathcal{V}ar(l, t_1, \ldots, t_{i-1})$. Note that a normal 1-CTRS is by definition also a DCTRS. A DCTRS $\mathcal{R}$ is *right-stable* (RS, cf. [18]) if for every conditional rule $l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$ we have $\mathcal{V}ar(l, s_1, t_1, \ldots, s_{i-1}, t_{i-1}, s_i) \cap \mathcal{V}ar(t_i) = \emptyset$, and t_i is a linear constructor term or a ground $\mathcal{R}_u$-normal form for all $1 \leq i \leq n$. To simplify the presentation of some results we will sometimes denote conditional rules as $t_0 \to s_{n+1} \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$.

The rewrite relation of an oriented CTRS $\mathcal{R}$ is inductively defined as follows: $R_0 = \emptyset$, $R_{j+1} = \{l\sigma \to r\sigma \mid l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n \in \mathcal{R} \wedge s_i\sigma \to^*_{R_j} t_i\sigma \text{ for all } 1 \leq i \leq n\}$, and $\to_{\mathcal{R}} = \bigcup_{j \geq 0} \to_{R_j}$.

2.2 Unravelings

There exists abundant literature on transforming CTRSs into unconditional systems. For a unified parameterized approach to such transformations and the relevant terminology we refer to [7]. Unravelings as introduced and investigated in [10] are the most simple and intuitive ones. We present here a sequential version of unraveling for DCTRS.

▶ **Definition 2.1** (sequential unraveling of DCTRSs, [16]). Let $\mathcal{R}$ be a DCTRS. For every conditional rule $\alpha\colon l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$ we use n new function symbols U_i^α ($i \in \{1, \ldots, n\}$). Then α is transformed into a set of unconditional rules as follows:[2]

$$\mathbb{U}_{seq}(\alpha) = \{l \to U_1^\alpha(s_1, \vec{X_1}) \ , \ U_1^\alpha(t_1, \vec{X_1}) \to U_2^\alpha(s_2, \vec{X_2}) \ , \ \ldots \ , \ U_n^\alpha(t_n, \vec{X_n}) \to r\}$$

where $X_i = \mathcal{V}ar(l, t_1, \ldots, t_{i-1})$. Any unconditional rule β of $\mathcal{R}$ is transformed into itself: $\mathbb{U}_{seq}(\beta) = \{\beta\}$. The transformed system $\mathbb{U}_{seq}(\mathcal{R}) = \mathcal{R}'_{seq} = (\mathcal{F}', R')$ is obtained by transforming each rule of $\mathcal{R}$ where $\mathcal{F}'$ is $\mathcal{F}$ extended by all new function symbols.

The *simultaneous* unraveling of a rule α from $\mathcal{R}$ as above just yields one introduction rule and one elimination rule: $\mathbb{U}_{sim}(\alpha) = \{l \to U^\alpha(s_1, \ldots, s_n, \vec{X}) \ , \ U^\alpha(t_1, \ldots, t_n, \vec{X}) \to r\}$ where $X = \mathcal{V}ar(l, t_1, \ldots, t_{n-1})$.

Unconditional rules remain invariant. The resulting (*unraveled*) TRS is denoted by $\mathbb{U}_{sim}(\mathcal{R})$ or $\mathcal{R}'_{sim}$.

▶ Remark (simultaneous versus sequential unraveling). Note that for a normal 1-CTRS $\mathcal{R}$, $\mathbb{U}_{sim}(\mathcal{R})$ is indeed a TRS. However, for a given DCTRS $\mathcal{R}$ its simultaneously unraveled system $\mathbb{U}_{sim}(\mathcal{R})$ does in general not satisfy the variable condition of TRSs $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ for its rules $l \to r$.

In [15], an "optimized" version $\mathbb{U}_{opt}$ of $\mathbb{U}_{seq}$ is presented. In $\mathbb{U}_{opt}$ variable bindings are only passed along the computation process if they are eventually used again. Intermediate results in other variables bindings are dropped. The conditional rule α is thereby transformed into $\mathbb{U}_{opt}(\alpha) = \{l \to U_1^\alpha(s_1, \vec{X_1}) \ , \ U_1^\alpha(t_1, \vec{X_1}) \to U_2^\alpha(s_2, \vec{X_2}) \ , \ \ldots \ , \ U_n^\alpha(t_n, \vec{X_n}) \to r\}$ where $X_i = \mathcal{V}ar(l, t_1, \ldots, t_{i-1}) \cap \mathcal{V}ar(t_i, s_{i+1}, t_{i+1}, \ldots, s_n, t_n, r)$.

Symbols from $\mathcal{F}' \setminus \mathcal{F}$ are also called U-symbols. Terms rooted by such symbols are called U-terms or U-rooted terms. Terms containing U-terms are *mixed terms* (as opposed

[2] Using $\overrightarrow{\mathcal{V}ar(s)}$ as sequence of the variables in s goes back to [16], whereas in [10] the sequence is constructed from the *multiset* of variables in s. The former version appears to be generally preferable, because it is more abstract and avoids additional complications due to "non-synchronization effects".

to *original terms*). Every U-symbol corresponds to a particular conditional rewrite rule of the original CTRS according to Definition 2.1. Hence, we write U_j^α to indicate that U_j^α corresponds to the rewrite rule α. Rewrite rules $l \to r$ of an unraveled DCTRS are called *original unconditional rules* if neither $\mathsf{root}(l)$ nor $\mathsf{root}(r)$ is a U-symbol, *U-introduction rules* if $\mathsf{root}(l)$ is not a U-symbol and $\mathsf{root}(R)$ is a U-symbol, *U-switch rules* (or just *switch rules*) if both $\mathsf{root}(l)$ and $\mathsf{root}(r)$ are U-symbols and *U-elimination rules* if $\mathsf{root}(l)$ is a U-symbol and $\mathsf{root}(r)$ is not a U-symbol.

If a property $\mathcal{P}$ is satisfied in the transformed DCTRS $\mathbb{U}_{seq}(\mathcal{R})$ ($\mathbb{U}_{opt}(\mathcal{R})$) then $\mathcal{R}$ satisfies the *ultra-property*, *ultra-$\mathcal{P}$* w.r.t. $\mathbb{U}_{seq}$ ($\mathbb{U}_{opt}$) (cf. [10]) or short $\mathbb{U}$-$\mathcal{P}$ ($\mathbb{U}_{opt}$-$\mathcal{P}$). Observe, that $\mathbb{U}_{opt}$-$\mathcal{P}$ is in many cases different from $\mathbb{U}$-$\mathcal{P}$. While $\mathbb{U}$-LL is equivalent to $\mathbb{U}_{opt}$-LL, $\mathbb{U}_{opt}$-RL and $\mathbb{U}_{opt}$-NE are more general than $\mathbb{U}_{seq}$-RL and $\mathbb{U}_{seq}$-NE, respectively.

From now on, unless stated otherwise, $\mathbb{U}$ is the unraveling $\mathbb{U}_{seq}$, $\mathcal{R} = (\mathcal{F}, R)$ is a DCTRS and $\mathcal{R}' = (\mathcal{F}', R')$ denotes its unraveled TRS (using $\mathbb{U}$). By $\mathcal{T}$ we mean the terms over the original signature $\mathcal{F}$ and by $\mathcal{T}'$ the terms over the extended signature $\mathcal{F}'$.

For proof-technical reasons, in particular in order to show that unraveled systems are not too general and do not enable "too many" reductions, we use a function that maps mixed terms to original terms. The idea of this function is to recursively substitute for each U-term the left-hand side of the corresponding conditional rule instantiated by the substitution which is determined by the variable bindings stored in the U-term.

▶ **Definition 2.2** (translate backwards (tb)). Let $\mathcal{R} = (\mathcal{F}, R)$ be a DCTRS. The mapping $\mathsf{tb} : \mathcal{T}' \to \mathcal{T}$ (read "translate back") which is equivalent to Ohlebusch's mapping ∇ ([16, Definition 7.2.53]) is defined as follows:

$$\mathsf{tb}(t) = \begin{cases} x & \text{if } t = x \in \mathcal{V} \\ f(\mathsf{tb}(t_1), \ldots, \mathsf{tb}(t_{\mathsf{ar}(f)})) & \text{if } t = f(t_1, \ldots, t_{\mathsf{ar}(f)}) \text{ and } f \in \mathcal{F} \\ l\sigma & \text{if } t = U_j^\alpha(u, v_1, \ldots, v_k) \text{ and } x_i\sigma = \mathsf{tb}(v_i) \text{ for } 1 \le i \le k \end{cases}$$

where α is the rule $l \to r \Leftarrow c$ and $\overrightarrow{\mathcal{V}ar(l)} = x_1, \ldots, x_{k'}$ $(1 \le k' \le k)$.

Observe, that tb cannot be sensibly defined for $\mathbb{U}_{opt}$, since in $\mathbb{U}_{opt}$ not all variable bindings of the left-hand side of rules are preserved in U-terms.

In this paper we focus on the property of *soundness* of unravelings which is dual to the (easier to obtain) property of completeness. An unraveling is said to be *complete for reduction* (or *simulation-complete*) for a class of CTRSs if for every CTRS $\mathcal{R}$ of this class, $u \to_{\mathcal{R}}^* v$ for $u, v \in \mathcal{T}$ implies $u \to_{\mathcal{R}'}^* v$. An unraveling is *sound for reduction* (or *simulation-sound*) if $u \to_{\mathcal{R}'}^* v$ implies $u \to_{\mathcal{R}}^* v$. We use the notion of *soundness for reduction to normal form*, which means that $u \to_{\mathcal{R}'}^* v$ with v being a normal form implies $u \to_{\mathcal{R}}^* v$. Given a particular CTRS $\mathcal{R}$, we also say that the unraveling is complete (sound) for $\mathcal{R}$ or, slightly abusing terminology, that $\mathcal{R}'$ is complete (sound) w.r.t. $\mathcal{R}$. For a more thorough discussion of the terminology used for (preservation properties of) transformations we refer to [7].

3 Sufficient Criteria for Soundness of Unraveling DCTRSs

For the case of a normal 1-CTRS $\mathcal{R}$ it was shown in [8] that $\mathcal{R}$ can soundly be (simultaneously) unraveled provided that it is either confluent, or non-erasing, or weakly left-linear or contains only ground conditions. A careful inspection of the proofs in [8] reveals that when using sequential instead of simultaneous unraveling the same results can also be proved in essentially the same way.

The main additional complication is here that when analyzing reduction sequences in the sequential unraveling case one now has not just one introduction and one elimination step for $\alpha\colon l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$, but also $n-1$ intermediate switch rule steps of the shape $U_i^\alpha(t_i, \vec{X}_i) \to U_{i+1}^\alpha(s_{i+1}, \vec{X}_{i+1})$.

When considering general DCTRSs instead of normal 1-CTRSs, there are two major sources of complication. First, right-hand sides of conditions of rules in DCTRSs need not be ground normal forms, and second, these right-hand sides may introduce extra variables that do not occur in the left-hand side of the conditional rule. Both of these properties indeed cause unsoundness in general even for e.g. confluent and $\mathbb{U}$-NE systems (cf. Example 3.1 below). Thus, it is necessary to restrict our results to certain classes of DCTRSs (cf. e.g. Theorems 3.6, 3.11 and 3.16 below).

3.1 Negative Results

Let us consider potential criteria for soundness of unraveling DCTRSs, namely confluent (CR) DCTRSs and DCTRSs $\mathcal{R}$ where $\mathcal{R}'$ is non-erasing (i.e., $\mathcal{R}$ is $\mathbb{U}$-NE). Unfortunately, both of these criteria do not extend to DCTRSs.

▶ **Example 3.1.** Consider the $\mathbb{U}$-NE and RS DCTRS $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3$ where

$$\mathcal{R}_1 = \left\{ \begin{tikzcd} a \arrow[r] \arrow[dr] & c & s(c) \arrow[r] \arrow[dr] & t(k) \\ b \arrow[r] \arrow[ur] & d & & t(l) \end{tikzcd} \right\} \qquad \begin{array}{l} \mathcal{R}_2 = \{g(x,x) \to h(x,x)\} \\ \\ \mathcal{R}_3 = \{f(x) \to \langle x, y\rangle \Leftarrow s(x) \to^* t(y)\} \end{array}$$

Unraveling of $\mathcal{R}$ yields $\mathcal{R}' = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \{f(x) \to U_1^\alpha(s(x), x)\ ,\ U_1^\alpha(t(y), x) \to \langle x, y\rangle\}$. $\mathcal{R}'$ gives rise to the derivation

$$\begin{aligned} g(f(a), f(b)) &\mathrel{+\!\!\!\!\!\!\Vert\!\!\!\to} g(U_1^\alpha(s(a), a), U_1^\alpha(s(b), b)) \to^* g(U_1^\alpha(s(c), d), U_1^\alpha(s(c), d)) \\ &\to h(U_1^\alpha(s(c), d), U_1^\alpha(s(c), d)) \to^* h(U_1^\alpha(t(k), d), U_1^\alpha(t(l), d)) \to^* h(\langle d, k\rangle, \langle d, l\rangle)\,. \end{aligned}$$

However, to get this reduction $g(f(a), f(b)) \to^* h(\langle d, k\rangle, \langle d, l\rangle)$ in $\mathcal{R}$, we need a term $v \in \mathcal{T}$ such that (1) $s(v) \to_{\mathcal{R}}^* t(w)$ where $w \in \{k, l\}$, (2) $a \to_{\mathcal{R}}^* v$ and $b \to_{\mathcal{R}}^* v$, and (3) $v \to_{\mathcal{R}}^* d$. By (1) and (2) $v = c$, yet this contradicts (3).

The DCTRS $\mathcal{R} \cup \{\ c \to e \leftarrow d\ ,\ k \to e \leftarrow l\ ,\ s(e) \to t(e)\ \}$ is even confluent. Note that this DCTRS is even operationally terminating (or, equivalently quasi-decreasing), cf. [6, 17]. Although $h(\langle d, k\rangle, \langle d, l\rangle)$ can now be reduced to $h(\langle e, e\rangle, \langle e, e\rangle)$ and $g(f(a), f(b)) \to_{\mathcal{R}}^* h(\langle e, e\rangle, \langle e, e\rangle)$, still $g(f(a), f(b)) \not\to_{\mathcal{R}}^* h(\langle d, k\rangle, \langle d, l\rangle)$ by the same argument as above.

Example 3.1 shows that neither confluence nor $\mathbb{U}$-NE are sufficient for soundness of unraveling DCTRSs, even if the system is right-stable and terminating.

For NE, the reason why an extension of the proof approach of [8] to (RS) DCTRSs is not possible in the general case, lies in the fact that for the construction in [8] we used a "translation forward" tf from reductions in the unraveled system to reductions in the original system. From an intermediate stage of evaluating the conditions of the unraveled version of some rule $l \to r \Leftarrow c$ we would need to calculate the final substitution for the right-hand side. However, this is in general impossible due to the incremental left-to-right computation character in DCTRSs. In fact, in Example 3.1, from the term $U_1^\alpha(s(c), c))$ (with $x = c$) we get two possible instances for y, namely k and l. Therefore tf cannot be sensibly defined here.

3.2 Confluence

Considering again Example 3.1 (the confluent version), we see that in the unsound $\mathcal{R}'$-reduction $g(f(a), f(b)) \to^*_{\mathcal{R}'} h(\langle d, k\rangle, \langle d, l\rangle)$ the result is not a normal form. If we only consider reduction from original terms to original terms which are normal forms then we can indeed guarantee soundness in the case of confluent right-stable DCTRSs. The proof works similar to the one for confluent normal 1-CTRSs, however, now instead of normality we exploit the fact that the final result of the reduction considered is a normal form.

Due to the specific structure of rules in unraveled DCTRSs, whenever there is an $\mathcal{R}'$ reduction $D\colon s \to^* U^\alpha_j(v, x_1, \ldots, x_n)\sigma$ where $s \in \mathcal{T}$, then intuitively the reduction sequences $s_i\sigma_i \to t_i\sigma_{i+1}$ that explicitly satisfy the conditions $s_i \to^* t_i$ for all $1 \leq i < j$ of the conditional rule $\alpha\colon l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$ must have occurred as subreductions of D (for some substitutions σ_i where $x\sigma_i \to^*_{\mathcal{R}'} x\sigma_{i+1}$ for all i and all x). This observation is formalized in the following lemma (which is used e.g. in the proof of Lemma 3.4 below).

▶ **Lemma 3.2** (extraction of condition evaluation). *Let $\mathcal{R}$ be a DCTRS. If $u_1 \to_{\mathcal{R}'} u_2 \to_{\mathcal{R}'} \cdots \to_{\mathcal{R}'} u_n \to_{\mathcal{R}'} v$ ($u_1 \in \mathcal{T}$), $v|_p = U^\alpha_j(v_1, \ldots, v_m)$ for some position p and $\alpha = l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$, then there exist substitutions $\sigma_1, \ldots, \sigma_j$ such that*

- *$s_i\sigma_i \to^*_{\mathcal{R}'} t_i\sigma_{i+1}$ for all $1 \leq i < j$, $s_j\sigma_j \to^*_{\mathcal{R}'} v_1$; and*
- *$x \in \mathcal{D}om(\sigma_i) \cap \mathcal{D}om(\sigma_{i+1})$ implies $x\sigma_i \to^*_{\mathcal{R}'} x\sigma_{i+1}$ for all $1 \leq i < j$; and*
- *if $U^\alpha_j(t_j, x_2, \ldots, x_m)$ is the left-hand side of the unique rule of $\mathcal{R}'$ defining U^α_j, then $x_i\sigma_j \to^*_{\mathcal{R}'} v_i$ for all $2 \leq i \leq m$.*

Moreover, for each single reduction step $s \to_{\mathcal{R}'} t$ in the reductions $s_i\sigma_i \to^*_{\mathcal{R}'} t_i\sigma_{i+1}$, $s_j\sigma_j \to^*_{\mathcal{R}'} v_1$, $x\sigma_i \to^*_{\mathcal{R}'} x\sigma_{i+1}$ and $x_i\sigma_j \to^*_{\mathcal{R}'} v_i$, there exists an index $k \leq n$ and a position q such that $u_k|_q = s$ and $u_{k+1}|_q = t$.

Proof. Straightforward by induction on the length of D. ◀

For the case of normal 1-CTRSs, confluence of a DCTRS $\mathcal{R}$ is a sufficient criterion for soundness of $\mathcal{R}'$ (cf. [8]). Normality of right hand-sides of conditions is vital for this result, because it means that if a conditional rule $\alpha\colon l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$ is applicable to a term $l\sigma \in \mathcal{T}$, then it is also applicable to a term $l\sigma' \in \mathcal{T}$ if $x\sigma \to^*_{\mathcal{R}} x\sigma'$ for all variables x. This is because applicability of α to $l\sigma$ means that $s_i\sigma \to^*_{\mathcal{R}} t_i$ for all $1 \leq i \leq n$. Moreover, we have $s_i\sigma \to^*_{\mathcal{R}} s_i\sigma'$ and thus by confluence and normality of t_i we obtain $s_i\sigma' \to^*_{\mathcal{R}} t_i$ for all $1 \leq i \leq n$.

This observation leads to the conjecture that confluence of a *right-stable* DCTRS is sufficient for soundness of $\mathcal{R}'$, because for right-stable DCTRSs right-hand sides of conditions are either ground normal forms or constructor terms with fresh variables. Hence, we have $s_i\sigma' \to t_i\theta'_i$ for all $1 \leq i \leq n$ and some substitution θ'_i provided that $l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$ is a right-stable rewrite rule, $s_i\sigma \to^*_{\mathcal{R}} t_i\theta$ for all $1 \leq i \leq n$ and $x\sigma \to^*_{\mathcal{R}} x\sigma'$ for all variables x. Indeed, as the following lemmas and Theorem 3.6 below show, $\mathcal{R}'$ is sound w.r.t. reductions to normal forms, provided that $\mathcal{R}$ is confluent and right-stable.

The following lemma states a monotonicity property of tb w.r.t. joining reductions. It is comparable to Lemma 3.8 in [8].

▶ **Lemma 3.3.** *Let $\mathcal{R} = (\mathcal{F}, R)$ be a DCTRS. If $s \to_{p,\mathcal{R}'} t$ and $\mathsf{tb}(s|_p) \downarrow_{\mathcal{R}} \mathsf{tb}(t|_p)$, then $\mathsf{tb}(s|_q) \downarrow_{\mathcal{R}} \mathsf{tb}(t|_{q'})$ for every descendant $t|_{q'}$ of $s|_q$.*

Proof Sketch. For the interesting case of $p \leq p_i$ we use induction on the length of q where $p.q = p_i$. ◀

The next lemma is the key lemma for proving soundness of $\mathcal{R}'$ for reduction to normal form of confluent right-stable DCTRSs.

▶ **Lemma 3.4** (technical key lemma). *Let $\mathcal{R} = (\mathcal{F}, R)$ be a confluent right-stable* DCTRS *and let $D\colon u_1 \to_{p_1,\mathcal{R}'} u_2 \to_{p_2,\mathcal{R}'} \dots \to_{p_{n-1},\mathcal{R}'} u_n$ be a $\mathcal{R}'$ reduction sequence where $u_1 \in \mathcal{T}$ and $u_i \in \mathcal{T}'$ for all $1 \leq i \leq n$. Then $\mathsf{tb}(u_i|_{p_i}) \downarrow_{\mathcal{R}} \mathsf{tb}(u_{i+1}|_{p_i})$ for all $1 \leq i < n$.*

Proof Sketch. Proof by induction on the length of D and case distinction on the rule applied in the last step of D. ◀

▶ **Lemma 3.5.** *Let $\mathcal{R} = (\mathcal{F}, R)$ be a confluent right-stable* DCTRS *and let $D\colon u_1 \to_{p_1,\mathcal{R}'} u_2 \to_{p_2,\mathcal{R}'} \dots \to_{p_{n-1},\mathcal{R}'} u_n$ be a $\mathcal{R}'$ reduction sequence where $u_1, u_n \in \mathcal{T}$, $u_i \in \mathcal{T}'$ for all $1 \leq i \leq n$ and u_n is a normal form. Then $u_1 \to^*_{\mathcal{R}} \mathsf{tb}(u_n)$.*

Proof. By Lemma 3.4, $\mathsf{tb}(u_i|_{p_i}) \downarrow_{\mathcal{R}} \mathsf{tb}(u_{i+1}|_{p_i})$ for all $i \in \{1, \dots, n-1\}$. By Lemma 3.3, $\mathsf{tb}(u_i|_{\epsilon}) \downarrow_{\mathcal{R}} \mathsf{tb}(u_{i+1}|_{\epsilon})$. Since u_n is a normal form and by confluence of $\mathcal{R}$ this implies $u_1 \to^*_{\mathcal{R}} \mathsf{tb}(u_n)$. ◀

▶ **Theorem 3.6.** *Let $\mathcal{R}$ be a confluent right-stable* DCTRS*. Then $\mathbb{U}_{seq}$ is sound for reduction to normal form.*

Proof. Straightforward by Lemma 3.5. ◀

Note that several results for verifying confluence of DCTRSs exist, cf. e.g. [18, 1].

▶ **Example 3.7.** Consider the DCTRS $\mathcal{R} = \mathcal{R}_{div} \cup \mathcal{R}_{sub} \cup \mathcal{R}_{leq}$ where

$$\mathcal{R}_{div} = \left\{ \begin{array}{c} div(0, s(y)) \to (0,0) \quad div(s(x), s(y)) \to \langle 0, s(x)\rangle \Leftarrow leq(s(y), s(x)) \to^* false \\ div(s(x), s(y)) \to \langle s(q), r\rangle \Leftarrow leq(s(y), s(x)) \to^* true, div(x-y, s(y)) \to^* \langle q, r\rangle \end{array} \right\}$$

$$\mathcal{R}_{sub} = \{ \quad s(x) - 0 \to s(x) \qquad 0 - s(y) \to 0 \qquad s(x) - s(y) \to x - y \quad \}$$

$$\mathcal{R}_{leq} = \{ \; leq(s(x), 0) \to false \quad leq(0, s(y)) \to true \quad leq(s(x), s(y)) \to leq(x, y) \; \}$$

performing a simple division with remainder. Transforming the conditional rules yields

$$\begin{array}{ll} div(s(x), s(y)) \to U_1^{\alpha_1}(leq(s(y), s(x)), x, y) & div(s(x), s(y)) \to U_1^{\alpha_2}(leq(s(y), s(x)), x, y) \\ U_1^{\alpha_1}(false, x, y) \to \langle 0, s(x)\rangle & U_1^{\alpha_2}(true, x, y) \to U_2^{\alpha_2}(div(x-y, s(y)), x, y) \\ & U_2^{\alpha_2}(\langle q, r\rangle, x, y) \to \langle s(q), r\rangle \end{array}$$

$\mathcal{R}$ is right-stable, left-linear, quasi-reductive (cf. [1, 17]), strongly deterministic ([1]) and has infeasible critical pairs. Thus it is locally confluent and confluent (using either [18] or [1]). Hence, by Theorem 3.6 we obtain that $\mathcal{R}'$ is sound for reduction to normal form.

3.3 Right-Linearity

In the case of normal 1-CTRSs $\mathcal{R}$ it was shown in [8] that right-linearity of $\mathcal{R}'$ implies that all left-hand sides of conditions are ground terms. Moreover, since right-hand sides of conditions in normal 1-CTRSs are ground terms as well, such systems are of limited practical interest. For the case of DCTRSs, the situation is slightly more interesting, since, while left-hand sides of conditions must still be ground terms in order to guarantee right-linearity of the unraveled unconditional system, right-hand sides of conditions may contain variables. In this section we show that for a DCTRS $\mathcal{R}$, right-linearity of $\mathcal{R}'$ implies soundness of $\mathcal{R}'$.

▶ **Lemma 3.8.** *Let $\mathcal{R} = (\mathcal{F}, R)$ be a* DCTRS*. If $s \to_{p,\mathcal{R}'} t$ and $\mathsf{tb}(s|_p) \to^*_{\mathcal{R}} \mathsf{tb}(t|_p)$, then $\mathsf{tb}(s|_q) \to^*_{\mathcal{R}} \mathsf{tb}(t|_{q'})$ for every descendant $t|_{q'}$ of $s|_q$.*

Proof. Analogous to Lemma 3.8 in [8]. ◀

▶ **Lemma 3.9.** *Let $\mathcal{R} = (\mathcal{F}, R)$ be a* DCTRS *such that $\mathcal{R}'$ is right-linear and let $D\colon u_1 \to_{p_1,\mathcal{R}'} u_2 \to_{p_2,\mathcal{R}'} \dots \to_{p_{n-1},\mathcal{R}'} u_n$ be a $\mathcal{R}'$ reduction sequence where $u_1 \in \mathcal{T}$ and $u_i \in \mathcal{T}'$ for all $1 \le i \le n$. Then $\mathsf{tb}(u_i|_{p_i}) \to_{\mathcal{R}}^* \mathsf{tb}(u_{i+1}|_{p_i})$ for all $1 \le i < n$.*

Proof Sketch. Proof by induction on the length of D and case distinction on the rule applied in the first step of D. ◀

▶ **Lemma 3.10.** *Let $\mathcal{R} = (\mathcal{F}, R)$ be a* DCTRS *such that $\mathcal{R}'$ is right-linear and let $D\colon u_1 \to_{p_1,\mathcal{R}'} u_2 \to_{p_2,\mathcal{R}'} \dots \to_{p_{n-1},\mathcal{R}'} u_n$ be a $\mathcal{R}'$ reduction sequence where $u_1 \in \mathcal{T}$, $u_i \in \mathcal{T}'$ for all $1 \le i \le n$. Then $u_1 \to_{\mathcal{R}}^* \mathsf{tb}(u_n)$.*

Proof. By Lemmas 3.9 and 3.8 we obtain $u_1 = \mathsf{tb}(u_1) \to_{\mathcal{R}}^* \mathsf{tb}(u_2) \to_{\mathcal{R}}^* \dots \to_{\mathcal{R}}^* \mathsf{tb}(u_n)$. ◀

▶ **Theorem 3.11.** *Let $\mathcal{R}$ be a $\mathbb{U}_{seq}$-*RL DCTRS*. Then, $\mathbb{U}_{seq}$ is sound for reduction.*

Proof. Straightforward by Lemma 3.10. ◀

3.4 Non-Erasingness

In [8] we proved soundness of $\mathbb{U}_{sim}$ for NE normal 1-CTRSs. For our proof it was essential that U-terms are not erased but properly eliminated. In order to ensure this, $\mathbb{U}$-NE is sufficient (which is equivalent to NE for normal 1-CTRSs).

Yet, we cannot extend our result for normal 1-CTRSs to DCTRSs because we used a *translation forward* (tf) that cannot be defined in an appropriate way if the rhs of a rule contains extra variables. In order to prove soundness using tf we therefore restrict ourselves to 2-DCTRSs. Example 3.1 shows indeed that $\mathbb{U}_{seq}$-NE (and also $\mathbb{U}_{opt}$-NE) is not a sufficient criterion for soundness.

$\mathbb{U}_{opt}$-NE is more general than $\mathbb{U}$-NE and tf can be properly defined for $\mathbb{U}_{opt}$ for 2-DCTRSs. Hence, we show our result for $\mathbb{U}_{opt}$. Since $\mathbb{U}$ is sound for a DCTRS $\mathcal{R}$ if $\mathbb{U}_{opt}$ is (cf. [15, Theorem 4.19]), this also yields soundness of $\mathbb{U}$.

Rules that use the same variable in the lhs and the rhs of conditions can still be a source of unsoundness. A more thorough analysis of CTRSs containing such rules shows that the following property is sufficient to exclude such cases of unsoundness: $\mathcal{V}ar(t_i) \cap \mathcal{V}ar(t_0, \dots, t_{i-1}) = \emptyset$ for all conditional rules $\alpha : t_0 \to s_{n+1} \Leftarrow s_1 \to^* t_1, \dots, s_n \to^* t_n$. This property resembles right-stability but is slightly more general since it allows non-linear rhs's in conditions. In the following we will refer to DCTRSs with this property as *right-separated* DCTRSs.

To prove soundness of $\mathbb{U}_{opt}$ for $\mathbb{U}_{opt}$-NE, right-separated 2-DCTRSs we first define the function "translate forward" for $\mathbb{U}_{opt}$:

▶ **Definition 3.12** (translation forward). Let $\mathcal{R} = (\mathcal{F}, R)$ be a 2-DCTRS. The mapping $\mathsf{tf}\colon \mathcal{T}' \to \mathcal{T}$ is defined by

$$\mathsf{tf}(t) = \begin{cases} x & \text{if } t = x \in \mathcal{V} \\ f(\mathsf{tf}(t_1), \dots, \mathsf{tf}(t_{\mathsf{ar}(f)})) & \text{if } t = f(t_1, \dots, t_{\mathsf{ar}(f)}) \text{ and } f \in \mathcal{F} \\ r\sigma & \text{if } t = U_j^\alpha(u, v_1, \dots, v_k) \text{ and } x_i\sigma = \mathsf{tf}(v_i) \text{ for } 1 \le i \le k \end{cases}$$

where α is the rule $l \to r \Leftarrow s_1 \to^* t_1, \dots, s_n \to^* t_n$, $\vec{X_j} = x_1, \dots, x_k$ and $X_j = \mathcal{V}ar(l, t_1, \dots, t_{j-1}) \cap \mathcal{V}ar(t_j, s_{j+1}, t_{j+1}, \dots, s_n, t_n, r)$.

By the definition of $\mathbb{U}_{opt}$, X_j contains all variables occurring in r for 2-DCTRSs so that tf is well-defined. The proof now follows mainly the proof for NE 1-CTRSs in [8]:

▶ **Lemma 3.13** (monotony property of tf). *Let $\mathcal{R} = (\mathcal{F}, R)$ be a 2-DCTRS. If $s \to_{p,\mathcal{R}} t$ for $s, t \in \mathcal{T}'$ and $\mathsf{tf}(s|_p) \to^*_{\mathcal{R}} \mathsf{tf}(t|_p)$, then $\mathsf{tf}(s|_q) \to^*_{\mathcal{R}} \mathsf{tf}(t|_{q'})$ for every descendants $t|_{q'}$ of $s|_q$.*

Proof Sketch. Analogous to Lemma 3.13 in [8]. ◀

▶ **Lemma 3.14** (technical key result for $\mathbb{U}_{opt}$-NE right-separated 2-DCTRSs). *Let $\mathcal{R} = (\mathcal{F}, R)$ be an $\mathbb{U}_{opt}$-NE and right-separated 2-DCTRS and let $D : u_1 \to_{p_1,\mathcal{R}'} u_2 \to_{p_2,\mathcal{R}'} \cdots u_n$ be a derivation where $u_n \in \mathcal{T}$ and $u_1, \ldots, u_{n-1} \in \mathcal{T}'$, then $\mathsf{tf}(u_i|_{p_i}) \to^*_{\mathcal{R}} \mathsf{tf}(u_{i+1}|_{p_i})$ for all $i \in \{1, \ldots, n-1\}$.*

Proof Sketch. Proof by induction on n and case distinction on the rule applied in the first rewrite step. The interesting case is where the first rewrite step is an introduction step of a conditional rule $\alpha : t_0 \to s_{n+1} \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$. Since $\mathcal{R}'$ is NE, all U-terms are eventually eliminated. By the induction hypothesis and Lemma 3.13, we inductively find matchers such that all conditions are satisfied. Because of right-separateness there are no conflicts with extra variables in the rhs of conditions. ◀

▶ **Lemma 3.15.** *Let $\mathcal{R}$ be a $\mathbb{U}_{opt}$-NE, right-separated 2-DCTRS, and $D : u_1 \to_{p_1,\mathcal{R}'} u_2 \to_{p_2,\mathcal{R}'} \cdots u_n$ be a derivation such that $u_n \in \mathcal{T}$ and $u_1, \ldots, u_{n-1} \in \mathcal{T}'$, then $\mathsf{tf}(u_1) \to^*_{\mathcal{R}} u_n$.*

Proof. By Lemma 3.14, $\mathsf{tf}(u_i|_{p_i}) \to^*_{\mathcal{R}} \mathsf{tf}(u_{i+1}|_{p_i})$ for all $i \in \{1, \ldots, n-1\}$. By Lemma 3.13, $\mathsf{tf}(u_i|_\epsilon) \to^*_{\mathcal{R}} \mathsf{tf}(u_{i+1}|_\epsilon)$. Since $\mathsf{tf}(u_n) = u_n$, $\mathsf{tf}(u_1) \to^*_{\mathcal{R}} u_n$. ◀

▶ **Theorem 3.16** (Soundness for reduction for $\mathbb{U}_{opt}$). *Let $\mathcal{R}$ be an $\mathbb{U}_{opt}$-NE and right-separated 2-DCTRS, then $\mathbb{U}_{opt}$ is sound for reduction.*

Proof. Straightforward via Lemma 3.15. ◀

3.5 Weak Left-Linearity

In [8] the class of weakly left-linear (WLL) normal 1-CTRSs has been introduced. Weakly left-linear normal 1-CTRSs may, in addition to left-linear rules, contain non-left-linear rules provided the variables occurring more than once in the the lhs of such a rule do not occur in its rhs at all. It was shown in [8] that for weakly left-linear normal 1-CTRSs $\mathbb{U}_{sim}$ is sound.

For a DCTRS $\mathcal{R}$ the situation is significantly more involved, since extra variables may occur in rhs's of multiple conditions of one conditional rewrite rule (which indeed can not be the case for normal 1-CTRSs). In this case, $\mathcal{R}'$ can be non-left-linear (indeed non-weakly-left-linear according to Definition [8, 3.22]) even if $\mathcal{R}$ is left-linear and the rhs's of all conditions are linear. Hence, WLL of a DCTRS $\mathcal{R}$ does not necessarily imply WLL of $\mathcal{R}'$ (and this is in sharp contrast to the situation of normal 1-CTRSs).

Hence, the notion of WLL from [8] is inadequate for our treatment of DCTRSs and we instead introduce and use a generalized notion of WLL. The basic idea of our definition of WLL for DCTRSs is to count simultaneous occurrences of variables in the rhs's of conditions and the lhs of a conditional rule, thus anticipating non-left-linear rules in the transformed systems. Variables that occur more than once in the lhs of a conditional rule and the rhs's of conditions should not occur at all in lhs's of conditions or the rhs of the conditional rule:

▶ **Definition 3.17** (Weakly left-linear DCTRSs). A DCTRS $\mathcal{R}$ is *weakly left-linear* (WLL) if for every rule $\alpha : t_0 \to s_{n_\alpha+1} \Leftarrow s_1 \to^* t_1, \ldots, s_{n_\alpha} \to^* t_{n_\alpha}$ in $\mathcal{R}$ and all variables $x \in \mathcal{V}ar(\alpha)$, $|t_0, \ldots, t_{n_\alpha}|_x > 1 \implies x \notin \mathcal{V}ar(s_1, \ldots, s_{n_\alpha+1})$.

Note that the version of weak left-linearity of [8] and the one from Definition 3.17 are compatible in the sense that WLL normal 1-CTRSs according to [8] are also WLL according to Definition 3.17. Hence, whenever we speak of WLL we mean the notion of Definition 3.17.

Given a WLL DCTRS $\mathcal{R}$, $\mathcal{R}'$ is not necessarily WLL (according to Definition 3.17). However, WLL of a DCTRS $\mathcal{R}$ does imply that in introduction or switch rules $l \to r$, every non-linear variable in l is linear in r, and elimination rules $l \to r$ erase all such variables.

In [8], we showed soundness of WLL normal 1-CTRSs by using a back translation w.r.t. derivations called tb_D to translate derivations in the transformed CTRS into derivations in the original CTRS. For DCTRSs $\mathcal{R}$, we are using a slightly modified version of tb_D that takes into account potential non-weak left-linearity of $\mathcal{R}'$:

▶ **Definition 3.18** (tb_D). Let $D : u_1 \to_{p_1,\mathcal{R}'} u_2 \to_{p_2,\mathcal{R}'} \cdots \to_{p_{n-1},\mathcal{R}'} u_n$ ($u_1 \in \mathcal{T}$), then tb_D is defined as

$$\mathsf{tb}_D(i,p) = \begin{cases} x & \text{if } u_i|_p = x \in \mathcal{V} \\ \mathsf{tb}_D(i-1,p') & \text{if } \mathsf{root}(u_i|_p) = U_j^\alpha \text{ and } p' \text{ is the unique one-step-ancestor of } p \\ f(\mathsf{tb}_D(i,p.1),\ldots,\mathsf{tb}_D(i,p.\mathsf{ar}(f)) & \text{if } \mathsf{root}(u_i|_p) = f, f \in \mathcal{F}, \text{ and} \\ & \mathsf{tb}_D(i,p.1),\ldots,\mathsf{tb}_D(i,p.\mathsf{ar}(f)) \text{ are defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

tb_D is undefined for U-terms with multiple one-step ancestors or terms containing such U-terms. This can happen if we apply a non-left-linear introduction or switch rule. After applying an introduction rule $l[x_i, x_i] \to U_1^\alpha(s_1, x_1, \ldots, x_k)$, tb_D would be undefined for U-terms in the $i+1$st argument in U_1^α-rooted terms. In the following, we will refer to such arguments in U-terms as *non-traceable arguments for* U_1^α.

Formally, the ith argument of the symbol U_j^α is a *non-traceable argument for* U_j^α, if in the introduction or switch rule $l \to U_j^\alpha(s_j, x_1, \ldots, x_k)$ in $\mathcal{R}'$, $|l|_{x_{i-1}} > 1$, or the rule is a switch rule, $l = U_{j-1}^\alpha(t_{j-1}, x_1, \ldots, x_{k'})$ and the ith argument of U_{j-1}^α is a *non-traceable argument for* U_{j-1}^α. If the ith argument of U_j^α ($2 \le i \le \mathsf{ar}(U_j^\alpha)$) is not a non-traceable argument, it is a *traceable argument (for* U_j^α*)*. Analogously, a term $u|_{p.i}$ ($i \in \mathbb{N}^+$) is a *(non-)traceable argument*, if $\mathsf{root}(u|_p) = U_j^\alpha$ and the ith argument of U_j^α is a (non-)traceable argument.

Our goal is to show that we can translate a derivation D of a transformed WLL DCTRS $\mathcal{R}'$ into a derivation of the original DCTRS using tb_D. Yet, tb_D might be undefined for terms occurring in D. To illustrate this problem, consider a (introduction or switch) rule $l[x, x] \to U_j^\alpha(s_j, x)$ (such that either $j = 1$, or x is not a non-traceable argument for U_{j-1}^α). Consider the derivation $l[u, u] \to_{\mathcal{R}'} U_j^\alpha(s, u) \to_{\mathcal{R}'} U_j^\alpha(s, v)$. tb_D is not defined for U-terms in u and v. We therefore cannot backtranslate the rewrite step $u \to v$ using tb_D.

Our basic idea to remedy this problem is to show that arbitrary $\mathcal{R}'$ reductions (starting from old terms) can be reconstructed into $\mathcal{R}'$ reductions (having the same starting and end term) such that no rewrite step occurs in a non-traceable argument (of any U-term). Intuitively, a derivation $l[u, u] \to U_j^\alpha(s, u) \to U_j^\alpha(s, v)$ is rebuilt by moving rewrite steps in non-traceable arguments ahead: $l[u, u] \mathrel{+\!\!\!\!\!\!\;\!\to} l[v, v] \to U_j^\alpha(s, v)$.

However, in general we cannot rebuild all derivations in WLL DCTRSs in this way: If a non-traceable argument has multiple descendants, we cannot move such rewrite steps ahead because they are not unique: Consider the rule $l'[z] \to r'[z, z]$ and the derivation $l'[l[u, u]] \to l'[U_j^\alpha(s, u)] \to r'[U_j^\alpha(s, u), U_j^\alpha(s, u)] \mathrel{+\!\!\!\!\!\!\;\!\to} r'[U_j^\alpha(s, v_1), U_j^\alpha(s, v_2)]$.

The rewrite steps $u \to v_1$ and $u \to v_2$ occur in non-traceable argument positions of U_j^α. In order to move these rewrite steps ahead, we first have to avoid rewrite steps that duplicate non-traceable arguments of U-terms. To achieve this, we further rearrange reduction sequences by moving applications of duplicating rewrite rules ahead of reductions in non-traceable arguments of involved U-terms. For instance, in a derivation $l'[l[u, u]] \to$

$l'[U_j^\alpha(s,u)] \to r'[U_j^\alpha(s,u), U_j^\alpha(s,u)]$ we shift the rewrite step multiplying the U-terms ahead in the following way: $l'[l[u,u]] \to r'[l[u,u], l[u,u]] \;\#\!\!\!\!\!\!\to\; r'[U_j^\alpha(s,u), U_j^\alpha(s,u)]$. Then we transform $r'[l[u,u], l[u,u]] \;\#\!\!\!\!\!\!\to\; r'[U_j^\alpha(s,u), U_j^\alpha(s,u)] \;\#\!\!\!\!\!\!\to\; r'[U_j^\alpha(s,v_1), U_j^\alpha(s,v_2)]$ into $r'[l[u,u], l[u,u]] \;\#\!\!\!\!\!\!\to\; r'[l[v_1,v_1], l[v_2,v_2]] \;\#\!\!\!\!\!\!\to\; r'[U_j^\alpha(s,v_1), U_j^\alpha(s,v_2)]$. Note that in the final reduction sequence no rewrite steps take place in non-traceable arguments of any U-term.

The following Lemma shows that we can transform a derivation $s[u] \to s[v] \to t[v,v]$ that duplicates a U-term containing a non-traceable argument into $s[u] \to t[u,u] \;\#\!\!\!\!\!\!\to\; t[v,v]$.

▶ **Lemma 3.19** (shifting ahead rewrite steps duplicating U-terms)**.** *Let $\mathcal{R}$ be a* WLL DCTRS*, $u, v, w \in \mathcal{T}'$ be such that $D : u \to_{p,\mathcal{R}'} v \to_{q,\mathcal{R}'} w$ where $v|_{p'.i}$ is a non-traceable argument, $p' \leq p$, either $q \parallel p'$ or $q < p'$, and such that there are no one-step descendants of $v|_{p'}$ inside a non-traceable argument. Then, there is a derivation $u \to_{\mathcal{R}'} w[u|_p]_P \;\#\!\!\!\!\!\!\to_{\mathcal{R}'}\; w[v|_p]_P = w$ where P is the set containing all one-step descendants of $v|_p$ in D.*

Proof Sketch. In the interesting case that $q < p'$, the rule $l \to r$ applied in the rewrite step $v \to w$ is linear for $v|_{p'}$. We therefore can apply $l \to r$ to $u|_q$. ◀

In the following lemma, we show that we can transform a derivation $s[u,u] \to t[u] \to t[v]$ into $s[u,u] \;\#\!\!\!\!\!\!\to\; s[v,v] \to t[v]$, thereby eliminating the rewrite step in the non-traceable argument in t.

▶ **Lemma 3.20** (shifting ahead rewrite steps in non-traceable arguments)**.** *Let $\mathcal{R}$ be a* WLL DCTRS*, $u, v, w \in \mathcal{T}'$ be such that D: $u \to_{p,\mathcal{R}'} v \to_{q,\mathcal{R}'} w$ where $v|_{q'.i}$ ($q'.i \leq q$) is a non-traceable argument and the only one-step-descendant of all its one-step-ancestors, and either $p < q'.i$ or $p \parallel q'.i$. Then, there is a derivation $u \;\#\!\!\!\!\!\!\to_{\mathcal{R}'}\; u[w|_q]_Q \to_{p,\mathcal{R}'} v[w|_q]_q = w$ where Q is the set containing all one-step ancestors of $v|_q$ in D.*

Proof Sketch. In the interesting case that $p < q'.i$, the rule $l \to r$ applied in the rewrite step $u \to v$ is right-linear for $v|_q$, so that can apply the rewrite step $v|_q \to w|_q$ to all one-step ancestors of $v|_q$, and then apply $l \to r$. ◀

By repeatedly applying these two Lemmata, we can eliminate all rewrite steps in non-traceable arguments:

▶ **Lemma 3.21** (elimination of rewrite steps in non-traceable arguments)**.** *Let $D : u_1 \to_{p_1,\mathcal{R}'} u_2 \to_{p_2,\mathcal{R}'} \cdots u_n$ be a derivation in $\mathcal{R}'$, then there is a derivation D': $u_1 = u_1' \to_{p_1',\mathcal{R}'} u_2' \to_{p_2',\mathcal{R}'} \cdots u_{n'}' = u_n$ such that there are no rewrite steps in non-traceable arguments.*

Proof Sketch. We prove this result by an inductive argument over the number of rewrite steps in non-traceable arguments in D: Assume the m^{th} rewrite step is the first one inside a non-traceable argument. We repeatedly apply Lemma 3.19 to the derivation $D : u_1 \to_{\mathcal{R}}^* u_{m+1}$ and obtain a new derivation $D' : u_1 \to_{\mathcal{R}'}^* u_m' \;\#\!\!\!\!\!\!\to_{\mathcal{R}'}\; u_{m+1}$ such that no non-traceable arguments are duplicated. We then can repeatedly apply Lemma 3.20 and thereby eliminate all rewrite steps in non-traceable arguments. ◀

The following example shows how we can rebuild a complex derivation such that it does not contain rewrite steps in non-traceable arguments:

▶ **Example 3.22.** Consider the WLL DCTRS $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ where

$$\mathcal{R}_1 = \{between(x,y,z) \to true \Leftarrow up(x) \to^* y, down(z) \to^* y\}$$

$$\mathcal{R}_2 = \left\{\begin{array}{lll} up(x) \to x & down(x) \to x & dup(x) \to \langle x, x\rangle \\ up(x) \to up(s(x)) & down(s(x)) \to down(x) & \end{array}\right\}$$

It is unraveled into $\mathcal{R}' = \left\{ \begin{array}{l} between(x,y,z) \to U_1^\alpha(up(x),x,y,z) \\ \quad U_1^\alpha(y,x,y,z) \to U_2^\alpha(down(z),x,y,z) \\ \quad U_2^\alpha(y,x,y,z) \to true \end{array} \right\} \cup \mathcal{R}_2$

We obtain the derivation

$$\begin{array}{ll} & dup(between(0,up(0),s(0))) \to dup(u_1(up(0),0,up(0),s(0))) \to \\ \to & dup(u_2(down(s(0)),0,up(0),s(0))) \to \\ \to & \langle u_2(down(s(0)),0,up(0),s(0)), u_2(down(s(0)),0,up(0),s(0))\rangle \to^* \\ \to^* & \langle u_2(down(s(0)),0,0,s(0)), u_2(down(s(0)),0,s(0),s(0))\rangle \end{array}$$

We want to move the rewrite steps in the non-traceable argument ahead of the application of the non-left-linear rule. Yet, the term $up(0)$ has two different descendants 0 and $s(0)$ in the last term of the derivation. In order to obtain a derivation without rewrite steps in non-traceable arguments, we move the introduction of the non-traceable argument behind the application of the duplicating rule (repeated application of Lemma 3.19). Then, we delay the introduction of the non-traceable argument (repeated application of Lemma 3.20):

$$\begin{tikzcd}
dup(between(0,up(0),s(0))) \arrow[d] \\
dup(u_1(up(0),0,up(0),s(0))) \arrow[d] \arrow[d, bend right] \\
dup(u_2(down(s(0)),0,up(0),s(0))) \arrow[d, shift right] \arrow[d, shift left] \\
\langle u_2(down(s(0)),0,up(0),s(0)), u_2(down(s(0)),0,up(0),s(0))\rangle \arrow[d, shift right] \arrow[d, shift left] \\
\langle u_2(down(s(0)),0,0,s(0)), u_2(down(s(0)),0,s(0),s(0))\rangle
\end{tikzcd}$$

$$\Longrightarrow$$

$$\begin{tikzcd}
dup(between(0,up(0),s(0))) \arrow[d, shift right] \arrow[d, shift left] \\
\langle between(0,up(0),s(0)), between(0,up(0),s(0))\rangle \arrow[d, shift right] \arrow[d, shift left] \\
\langle between(0,0,s(0)), between(0,s(0),s(0))\rangle \arrow[d, shift right] \arrow[d, shift left] \\
\langle u_1(up(0),0,0,s(0)), u_1(up(0),0,s(0),s(0))\rangle \arrow[d, shift right=3] \arrow[d, shift right] \arrow[d, shift left] \arrow[d, shift left=3] \\
\langle u_1(0,0,0,s(0)), u_1(s(0),0,s(0),s(0))\rangle \arrow[d, shift right=3] \arrow[d, shift right] \arrow[d, shift left] \arrow[d, shift left=3] \\
\langle u_2(down(s(0)),0,0,s(0)), u_2(down(s(0)),0,s(0),s(0))\rangle
\end{tikzcd}$$

After rebuilding the derivation, we can use the same proof structure we already used in [8], although we have to consider switch rules and take care that $\mathtt{tb}_D$ is still undefined for U-terms in non-traceable arguments. Lemma 3.25 shows that we can use $\mathtt{tb}$ for such cases.

First we show monotony of $\mathtt{tb}_D$:

▶ **Lemma 3.23** (monotony of $\mathtt{tb}_D$). *Let $D\colon u_1 \to_{p_1,\mathcal{R}'} u_2 \to_{p_2,\mathcal{R}'} \cdots \to_{p_{n-1},\mathcal{R}'} u_n$ be a derivation without rewrite steps in non-traceable arguments such that $\mathtt{tb}_D(i,p_i) \to^*_{\mathcal{R}} \mathtt{tb}_D(i+1,p_i)$ for all $i \in \{1,\dots,n-1\}$. If $u_i|_p$ ($p \in \mathcal{P}os(u_i)$) has a one-step-descendant $u_{i+1}|_{p'}$ and $\mathtt{tb}_D(i+1,p')$ is defined, then $\mathtt{tb}_D(i,p) \to^*_{\mathcal{R}} \mathtt{tb}_D(i+1,p')$.*

Proof Sketch. By case distinction on p and p_i. For the interesting case $p \leq p_i$, we use induction on the length of q, determined by $p.q = p_i$. ◀

The next auxiliary lemma helps us to extract derivations inside U-terms:

▶ **Lemma 3.24** (Extraction of U-terms). *Let $D\colon u_1 \to_{p_1,\mathcal{R}'} u_2 \to_{p_2,\mathcal{R}'} \cdots \to_{p_{n-1},\mathcal{R}'} u_n$ be a derivation without rewrite steps in non-traceable arguments such that $\mathtt{tb}_D(i,p_i) \to^*_{\mathcal{R}} \mathtt{tb}_D(i+1,p_i)$ for all $i \in \{1,\dots,n-1\}$. Let $u_m|_q = U_j^\alpha(v, x_1\tau,\dots,x_{|X_j|}\tau)$ such that $\mathtt{tb}_D(m,q)$ is defined, then (1) (extraction of [extra] variables) if $\mathtt{tb}_D(m,q.(i+1))$ is defined, then $\mathtt{tb}_D(m-1,q'_i) \to^*_{\mathcal{R}} \mathtt{tb}_D(m,q.(i+1))$ where q'_i is a one-step ancestor of $u_m|_{q.(i+1)} = x_i\tau$ ($1 \leq i \leq |X_j|$), and (2) (extraction of conditional arguments) if $u_{m-1}|_{q'} = U_j^\alpha(u, x_1\sigma,\dots,x_k\sigma)$, then $\mathtt{tb}_D(m-1,q'.1) \to^*_{\mathcal{R}} \mathtt{tb}_D(m,q.1)$.*

Proof. Straightforward via Lemma 3.23. ◀

Although we can prevent that there are rewrite steps applied in non-traceable arguments we still might obtain terms for which tb_D is undefined, namely terms in non-traceable arguments containing U-terms. For such terms we can use tb instead of tb_D as the following result shows.

▶ **Lemma 3.25** (tb_D to tb). *Let* $D\colon u_1 \to_{p_1,\mathcal{R}'} u_2 \to_{p_2,\mathcal{R}'} \cdots \to_{p_{n-1},\mathcal{R}'} u_n$ *be a derivation without rewrite steps in non-traceable arguments such that* $\mathsf{tb}_D(i,p_i) \to^*_{\mathcal{R}} \mathsf{tb}_D(i{+}1,p_i)$ *for all* $i \in \{1,\ldots,n-1\}$*. Then,* $\mathsf{tb}_D(m,q) \to^*_{\mathcal{R}} \mathsf{tb}(u_m|_q)$ *for all* $m \in \{1,\ldots,n\}$ *and all* $q \in \mathcal{P}os(u_m)$ *such that* $\mathsf{tb}_D(m,q)$ *is defined.*

Proof Sketch. By induction on the term depth of $u_m|_q$. In the interesting case that $u_m|_q$ is a U-term, $\mathsf{tb}_D(m,q) = \mathsf{lin}(l)\sigma$ and $\mathsf{tb}(u_m|_q) = l\tau$. By repeated application of Lemma 3.24 we obtain $x\sigma \to^*_{\mathcal{R}} x\tau$ for traceable arguments. If $\mathsf{tb}_D(m,q.i)$ is undefined (i.e., $u_m|_{q.i}$ is a non-traceable arguments), we exploit the fact that there are no rewrite steps in non-traceable arguments. We then obtain $\mathsf{tb}_D(k,q') \to^* \mathsf{tb}(u_k|_{q'}) = \mathsf{tb}(u_m|_{q.i})$ for ancestors $u_k|_{q'}$ of $u_m|_{q.i}$ such that $\mathsf{tb}_D(k,q')$ is defined. ◀

The following is our key result for WLL DCTRSs:

▶ **Lemma 3.26** (technical key result). *Let* $\mathcal{R}$ *be a* WLL DCTRS *and* $D: u_1 \to_{p_1,\mathcal{R}'} u_1 \to_{p_2,\mathcal{R}'} \cdots \to_{p_{n-1},\mathcal{R}'} u_n$ *(*$u_1 \in \mathcal{T}$*) be a derivation without rewrite steps in non-traceable arguments, then* $\mathsf{tb}_D(i,p_i) \to^*_{\mathcal{R}} \mathsf{tb}_D(i+1,p_i)$*.*

Proof Sketch. By induction on n and a case distinction on the type of rule applied in the last rewrite step. The interesting case is the application of the elimination rule of the conditional rule $t_0 \to^* s_{k+1} \Leftarrow s_1 \to^* t_1,\ldots,s_k \to^* t_k$. In this case $\mathsf{tb}_D(n-1,p_{n-1}) = \mathsf{lin}(t_0)\sigma_1$ and $\mathsf{tb}_D(n,p_{n-1}) = s_{k+1}\tau_k$. By Lemma 3.24 and the induction hypothesis we obtain derivations $s_i\tau_i \to^*_{\mathcal{R}} \mathsf{lin}(t_i)\sigma_i\sigma'_i$ where σ_i is the matcher for variables containing traceable arguments in t_i and σ'_i for variables containing non-traceable arguments. Observe, that $\mathcal{D}om(\tau_i) \cap \mathcal{D}om(\sigma'_j) = \emptyset$. By Lemma 3.24, $x\sigma_i \to^* x\tau_i$ so that $s_i\sigma_1\ldots\sigma_{i-1} \to^* s_i\tau_i$. By Lemma 3.25, $\mathsf{lin}(t_i)\sigma_{i+1}\sigma'_{i+1} \to^* t_i\sigma_{i+1}\tau'_{i+1}$ where τ'_{i+1} contains the tb-backtranslation of the non-traceable arguments in t_i (since there are no rewrite steps in non-traceable arguments, there are no rewrite steps in τ'_i). Let now $\sigma = \sigma_1\ldots\sigma_{k+1}\tau'_1\ldots\tau'_{k+1}$. Observe, that $s_i\sigma_1\ldots\sigma_i = s_i\sigma$ and $t_i\sigma_{i+1}\tau'_{i+1} = t_i\sigma$, so that $t_0\sigma \to_{\mathcal{R}} s_{k+1}\sigma$. Finally, repeated application of Lemma 3.24 yields $s_{k+1}\sigma \to^*_{\mathcal{R}} s_{k+1}\tau_{k+1}$. ◀

Finally, we obtain soundness for WLL DCTRSs:

▶ **Lemma 3.27** (Soundness for weakly left-linear DCTRSs). *Let* $\mathcal{R}$ *be a* WLL DCTRS *and* $D\colon u_1 \to_{p_1,\mathcal{R}'} u_1 \to_{p_2,\mathcal{R}'} \cdots \to_{p_{n-1},\mathcal{R}'} u_n$ *(*$u_1 \in \mathcal{T}$*) then* $u_1 \to^*_{\mathcal{R}} \mathsf{tb}(u_n)$*.*

Proof. By Lemma 3.21, there is a derivation $D'\colon u_1 \to^*_{\mathcal{R}'} u_n$ without rewrite steps in non-traceable arguments. In D', $\mathsf{tb}_D(n,\epsilon)$ is defined so that via Lemma 3.26 and repeated application of Lemma 3.23 we obtain $u_1 \to^*_{\mathcal{R}} \mathsf{tb}_D(n,\epsilon)$. By Lemma 3.25 we finally get $\mathsf{tb}_D(n,\epsilon) \to^*_{\mathcal{R}} \mathsf{tb}(u_n)$. ◀

▶ **Theorem 3.28** (Soundness for reduction of weakly left-linear DCTRSs). *The unraveling is sound for reduction w.r.t. weakly left-linear* DCTRSs*.*

Proof. Straightforward via Lemma 3.27. ◀

The definition of WLL DCTRSs allows conditional non-left-linear rules so that this result is more general than our result in [8] even for normal 1-CTRSs.

4 Discussion, Perspectives and Related Work

We have shown that the main ideas and approaches of [8] for soundness of normal 1-CTRSs also extend to DCTRSs, but with a couple of complications and subtleties. We think that especially the results involving weak left-linearity, non-erasingness and confluence as sufficient conditions for soundness (in the case of confluence restricted to soundness for reductions to normal forms) are quite interesting and practically relevant.

We are not aware of many related works on DCTRS or other classes of 3- or 4-CTRSs. An early one is [11] (which contains only a claim but no proofs). Yet, there is a notable exception. In [12] and, building on that paper, in [15] by the same authors closely related questions are investigated and similar soundness results for $\mathbb{U}_{opt}$ are obtained using quite different proof techniques.

More precisely, the main results of [15] are:

(a) $\mathbb{U}_{opt}$ is sound for $\mathbb{U}_{opt}$-LL (and for $\mathbb{U}$-LL) DCTRSs([15, Theorem 4.5]).

(b) $\mathbb{U}_{opt}$ is sound for $\mathbb{U}_{opt}$-RL-NE DCTRSs([15, Theorem 4.12]).

(c) If $\mathbb{U}_{opt}$ is sound w.r.t. $\mathcal{R}$, then $\mathbb{U}$ is so, too ([15, Theorem 4.19]).

Result (a) is of course important and practically relevant, since ($\mathbb{U}_{opt}$- or $\mathbb{U}$-)LL are reasonable requirements for DCTRSs. It is subsumed by our result on weak left-linearity (WLL) as sufficient criterion for soundness of $\mathbb{U}$ (Theorem 3.28). As Example 4.1 shows, $\mathbb{U}_{opt}$ is unsound even for WLL normal 1-CTRSs.

Result (b) resembles our Theorem 3.16, since it also requires $\mathbb{U}_{opt}$-NE. However, our result additionally forbids extra variables on rhs's. On the other hand, being right-separated is a much less restrictive property than $\mathbb{U}_{opt}$-RL. Proof-technically, the approach for (b) in [15] is remarkable and elegant. The proof works by reduction to the proof of (a) using the inverse reduction relation.

Result (c) is of particular interest, since via $\mathbb{U}_{opt}$ certain properties (e.g. soundness criteria) like NE become more likely to be satisfied. However, the reverse of (c) does not hold in general. In particular, we cannot reproduce our soundness results for WLL and CR (in the latter case w.r.t. soundness for reductions to normal forms) for $\mathbb{U}_{opt}$:

▶ **Example 4.1** (unsoundness of $\mathbb{U}_{opt}$). Consider the WLL and confluent normal 1-CTRS $\mathcal{R}$ consisting of the rules $or(x,y) \to true \Leftarrow x \to^* true$ and $eq(x,x) \to true$. $\mathcal{R}$ is unraveled using $\mathbb{U}_{opt}$ into $\mathcal{R}'_{opt} = \{or(x,y) \to U_1^{\alpha}(x)\ ,\ U_1^{\alpha}(true) \to true\ ,\ eq(x,x) \to true\}$.

In $\mathcal{R}'_{opt}$, $or(false, true)$ and $or(false, false)$ both rewrite to the irreducible U-term $U_1^{\alpha}(false)$, so that $eq(or(false, true), or(false, false)) \to^*_{\mathcal{R}'_{opt}} true$.

Yet, $or(false, true)$ and $or(false, false)$ are irreducible (and therefore not joinable) in $\mathcal{R}$ so that $eq(or(false, true), or(false, false)) \not\to^*_{\mathcal{R}} true$. Therefore, $\mathbb{U}_{opt}$ is not sound for $\mathcal{R}$.

In future work we will try to improve the (un)soundness analysis for DCTRSs (e.g. by isolating abstract principles or characterization results), apply the sondness criteria for deriving properties of the original systems, and exploit / transfer the analysis and criteria for other well-known transformation approaches from CTRSs to TRSs.

Acknowledgements: We are grateful to the anonymous reviewers for their detailed and helpful comments and criticisms!

References

1 J. Avenhaus and C. Loría-Sáenz. On conditional rewrite systems with extra variables and deterministic logic programs. In F. Pfenning, editor, *Proc. 5th LPAR, Kiev, Ukraine, July 1994*, pp. 215–229, 1994.

2 F. Baader and T. Nipkow. *Term rewriting and All That.* Cambridge Univ. Press, 1998.

3 M. Bezem, J. Klop, and R. Vrijer, editors. *Term Rewriting Systems.* Cambridge Tracts in Theoretical Computer Science 55. Cambridge University Press, Mar. 2003.

4 N. Dershowitz and D. Plaisted. Logic programming cum applicative programming. In *Proc. of the 1985 Symp. on Logic Programming, Boston, MA, July 1985*, pp. 54–66. IEEE, 1985.

5 F. Durán, S. Lucas, J. Meseguer, C. Marché, and X. Urbain. Proving termination of membership equational programs. In N. Heintze and P. Sestoft, eds., *PEPM'04*, pp. 147–158. ACM, 2004.

6 F. Durán, S. Lucas, J. Meseguer, C. Marché, and X. Urbain. Proving operational termination of membership equational programs. *Higher-Order and Symbolic Computation*, 21(10):59–88, 2008.

7 K. Gmeiner and B. Gramlich. Transformations of conditional rewrite systems revisited. In A. Corradini and U. Montanari, eds., *Recent Trends in Algebraic Development Techniques (WADT 2008) – Selected Papers*, LNCS 5486, pp. 166–186. Springer, 2009.

8 K. Gmeiner, B. Gramlich, and F. Schernhammer. On (un)soundness of unravelings. In C. Lynch, ed., *Proc. RTA, July 2010, Edinburgh, Scotland, UK*, LIPIcs (Leibniz International Proceedings in Informatics), 2010.

9 K. Gmeiner, B. Gramlich, and F. Schernhammer. On soundness conditions for unraveling deterministic conditional rewrite systems. Tech. Rep. E1852-2012-01, TU Wien, 2012. http://www.logic.at/staff/gramlich/papers/techrep-e1852-2012-01.pdf.

10 M. Marchiori. Unravelings and ultra-properties. In M. Hanus and M. M. Rodríguez-Artalejo, eds., *Proc. 5th ALP, Aachen*, LNCS 1139, pp. 107–121. Springer, 1996.

11 M. Marchiori. On deterministic conditional rewriting. Technical Report MIT LCS CSG Memo n.405, MIT, Cambridge, MA, USA, Oct. 1997.

12 N. Nishida, M. Sakai, , and T. Sakabe. On simulation-completeness of unraveling for conditional term rewriting systems. *IEICE Technical Report SS2004-18*, 104(243):25–30, 2004. Revised version from December 27, 2005, 15 pages.

13 N. Nishida and M. Sakai. Completion after program inversion of injective functions. *ENTCS* 237:39–56. Proc. 8th WRS, Hagenberg, Austria, July 2008, A. Middeldorp, ed., 2009

14 N. Nishida, M. Sakai, and T. Sakabe. Partial inversion of constructor term rewriting systems. In J. Giesl, ed., *Proc. 16th RTA, Nara, Japan, April 2005*, LNCS 3467, pp. 264–278. Springer, 2005.

15 N. Nishida, M. Sakai, and T. Sakabe. Soundness of unravelings for deterministic conditional term rewriting systems via ultra-properties related to linearity. In M. Schmidt-Schauss, ed., *Proc. 22nd RTA, May/June 2011, Novi Sad, Serbia*, LIPIcs (Leibniz International Proceedings in Informatics), pp. 267–282, 2011.

16 E. Ohlebusch. *Advanced Topics in Term Rewriting.* Springer, 2002.

17 F. Schernhammer and B. Gramlich. Characterizing and proving operational termination of deterministic conditional term rewriting systems. *Journal of Logic and Algebraic Programming*, 79(7):659–688, 2010.

18 T. Suzuki, A. Middeldorp, and T. Ida. Level-confluence of conditional rewrite systems with extra variables in right-hand sides. In J. Hsiang, ed., *Proc. 6th RTA, Kaiserslautern, Germany*, LNCS 914, pp. 179–193, Springer-Verlag. 1995.

19 P. Viry. Elimination of conditions. *J. Symb. Comput.*, 28(3):381–401, 1999.

Reinterpreting Compression in Infinitary Rewriting

Jeroen Ketema

Faculty EEMCS, University of Twente
PO Box 217, 7500 AE Enschede, the Netherlands
`j.ketema@ewi.utwente.nl`

Abstract

Departing from a computational interpretation of compression in infinitary rewriting, we view compression as a degenerate case of standardisation. The change in perspective comes about via two observations: (a) no compression property can be recovered for non-left-linear systems and (b) some standardisation procedures, as a 'side-effect', yield compressed reductions.

1998 ACM Subject Classification F.1.1 Models of Computation, F.3.2 Semantics of Programming Languages, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases term rewriting, infinitary rewriting, compression, standardisation

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.209

Category Regular Research Paper

1 Introduction

One of the most fundamental properties studied in infinitary rewriting is the so-called *compression property*. Roughly, the property states that for every reduction of transfinite length a 'similar' reduction can be found of length at most ω (the first infinite ordinal). Consider, e.g., the binary function symbol f and the rules $a \to g(a)$ and $b \to g(b)$. We have the following reduction of length $\omega + \omega$:

$$f(a,b) \to f(g(a),b) \to \cdots \to f(g^n(a),b) \to \cdots$$
$$f(g^\omega,b) \to f(g^\omega,g(b)) \to \cdots \to f(g^\omega,g^n(b)) \to \cdots f(g^\omega,g^\omega)\,.$$

By interleaving the a- and b-steps, we can compress this reduction to obtain a 'similar' reduction of length ω:

$$f(a,b) \to f(g(a),b) \to f(g(a),g(b)) \to \cdots \to f(g^n(a),g^n(b)) \to \cdots f(g^\omega,g^\omega)\,.$$

This second reduction has a very appealing property: We can obtain an arbitrarily good approximation of the final term by rewriting $f(a,b)$ a sufficient, *finite* number of times. As we are now in the realm of finite rewriting, it can be said that compression gives computational meaning to infinitary rewriting (see also [8], although room is left there for other interpretations than a computational one).

There are, however, two problems with the aforementioned computational interpretation. First, the compression property does not apply to all rewrite systems, while it can be argued that every rewrite system computes something. In particular, the property can fail for systems with non-left-linear rules. Consider, e.g., the non-left-linear rule $f(x,x) \to c$. This rule, in combination with the rules $a \to g(a)$ and $b \to g(b)$ from above, yields the standard counterexample to compression for non-left-linear systems [5, 8, 7]; the following reduction is of length $\omega + 1$ and cannot be compressed:

$$f(a,b) \to f(g(a),b) \to f(g(a),g(b)) \to \cdots \to f(g^n(a),g^n(b)) \to \cdots f(g^\omega,g^\omega) \to c\,.$$

Essentially, we need ω steps to obtain $f(g^\omega, g^\omega)$ before we can rewrite to c.

The second problem with the computational interpretation has to do with the fact that infinitary rewriting is susceptible to a similar computational treatment [10] as that of the real numbers in computable analysis [15]. We can think of both terms and reductions as Turing Machines. As such, any term along every transfinite reduction can be approximated with arbitrary precision in finite time (by executing the Turing Machines). Of course, such a computational treatment is not perfect: Although terms and reductions can be represented by Turning Machines, we cannot compute whether a reduction step that occurs in any such representation is indeed 'valid', as computing the validity of a match would take infinite time in case a non-left-linear rule is employed. Instead, with each representation of a reduction one should provide a 'certificate' (i.e. a proof) showing the validity of the matches that occur.

In the current paper we address both points of critique regarding the computational interpretation. To address the first, i.e. the lack of compression in non-left-linear systems, it would suffice if we could establish a *generalised* compression property: For each system a countable ordinal α might exist such that each reduction within that system can be compressed to one of length at most α [11, 12]. Unfortunately, as we will show, such a generalised compression property fails even for very simple systems. Hence, and as is also needed to address the second point of critique, a different interpretation of compression — one outside the realm of computability — is warranted for.

We reinterpret compression as a degenerate case of standardisation. Such a reinterpretation is not unexpected: Likewise to connections that exist between the equivalence of reductions and standardisation in finite rewriting [14], connections can be drawn between equivalence and compression in infinitary rewriting [8]. To enable our reinterpretation, we provide the first ever standardisation procedures for infinitary Term Rewriting Systems (iTRSs). We will show that these procedures, as a 'side-effect', yield compressed reductions.

The paper now proceeds as follows: In Section 2, we state a number of preliminaries needed in the remainder of the paper. We discuss the generalised compression property and its failure in Section 3. In Section 4, we formulate two standardisation procedures. Finally, in Section 5, we conclude.

2 Preliminaries

We briefly review some basic facts regarding infinitary Term Rewriting Systems (iTRSs); see [8, 7] for more detailed accounts. Throughout, we denote the first infinite ordinal by ω and arbitrary ordinals by α, β, γ, and so on; $[\alpha, \beta)$ denotes a left-closed, right-open interval of ordinals and $(\alpha, \beta]$ a right-open, left-closed interval. By $\mathbb{N}$ we denote the natural numbers including zero.

Terms. Let Σ be a signature, each element of which has finite arity, and let V be a countably infinite set of variables. The set of (finite and infinite) terms is commonly defined by metric completion [2, 8, 7]. Here, we give the shorter, but equivalent, definition from [3, 9].

▶ **Definition 2.1.** The set of *terms* $\mathcal{T}er(\Sigma, V)$ is defined coinductively such that x is a term for each $x \in V$ and if $f(t_1, \ldots, t_n)$ is a term, then $f \in \Sigma$ is n-ary and $t_1, \ldots, t_n$ are terms.

Substitutions over terms are defined by interpreting the usual definition coinductively [8, 7]. For the *root symbol*, $\mathrm{root}(t)$, of a term t we have $\mathrm{root}(x) = x$ and $\mathrm{root}(f(t_1, \ldots, t_n)) = f$.

The set of positions $\mathcal{P}os(t)$ of a term t is a set of *finite* strings over $\mathbb{N}$ representing the 'locations' of subterms in t [8, 7]. Denoting the empty string by ϵ, we have $\mathcal{P}os(x) = \{\epsilon\}$

and $\mathcal{P}os(f(t_1, \ldots, t_n)) = \{\epsilon\} \cup \bigcup_{1 \leq i \leq n} \{i \cdot p \mid p \in \mathcal{P}os(t_i)\}$. If p is a position of t, then $t|_p$ denotes the *subterm* of t at position p; we have $t|_\epsilon = t$ and $f(t_1, \ldots, t_i, \ldots, t_n)|_{i \cdot p} = t_i|_p$. By $t[s]_p$ we denote the *replacement* of the subterm at position p in t by s; we have $t[s]_\epsilon = s$ and $f(t_1, \ldots, t_i, \ldots, t_n)[s]_{i \cdot p} = f(t_1, \ldots, t_i[s]_p, \ldots, t_n)$. The *length* of p is denoted $|p|$. There exists a well-founded order $<$ on positions: $p < q$ iff p is a proper prefix of q. We write $\leq$ for the reflexive closure of $<$. If neither $p \leq q$ nor $q \leq p$, then p and q are *parallel* and we write $p \parallel q$. The concatenation of positions p and q is denoted by $p \cdot q$.

Rewrite Rules and Reductions. Rewrite rules and iTRSs are defined as in the finite case, except that the finiteness restriction on the right-hand side of rewrite rules is dropped:

▶ **Definition 2.2.** A *rewrite rule* is a pair of terms (l, r), denoted $l \to r$, with l finite and such all variables that occur in r also occur in l. A rewrite rule is *left-linear*, if each variable occurs at most once in l.

An *infinitary Term Rewriting System (iTRS)* is a pair $\mathcal{R} = (\Sigma, R)$ with Σ a signature and R a set of rewrite rules over Σ. An iTRS is *left-linear* if all its rewrite rules are.

Rewrite steps are now defined as usual:

▶ **Definition 2.3.** A *rewrite step* is a pair of terms $s \to t$ adorned with a position p and a rewrite rule $l \to r$ such that $s = s[\sigma(l)]_p$ and $t = s[\sigma(r)]_p$ for some substitution σ. The term $\sigma(l)$ is called an $l \to r$*-redex*. The redex *occurs* at position p and depth $|p|$ in s.

The previous gives sufficient background to define strongly convergent reductions [8, 7] (the most common notion of reduction in infinitary rewriting; see [6] for further discussion of notions of reduction in infinitary rewriting).

▶ **Definition 2.4.** A *strongly convergent reduction* of ordinal length α, denoted $t_0 \twoheadrightarrow^\alpha t_\alpha$, is a pair consisting of sequence of *terms* $(t_\beta)_{\beta < \alpha+1}$ and a sequence of *steps* $(p_\beta, l_\beta \to r_\beta)_{\beta < \alpha}$ such that for all $\beta < \alpha$ it holds that (a) $t_\beta \to t_{\beta+1}$ is a rewrite step adorned with the position p_β and the rewrite rule $l_\beta \to r_\beta$, and (b) if β is a limit ordinal, then t_γ converges to t_β and $|p_\gamma|$ tends to infinity when γ approaches β from below.

Here, a sequence of terms $(t_\gamma)_{\gamma < \beta}$ is said to converge to a term t_β whenever the depth up to which t_γ and t_β are identical increases as γ approaches β.

In case we are only interested in an upper bound, respectively a lower bound, α on the length of a strongly convergent reduction we write $s \twoheadrightarrow^{\leq \alpha} t$, respectively $s \twoheadrightarrow^{\geq \alpha} t$. Moreover, in case the length is irrelevant, respectively finite, we write $s \twoheadrightarrow t$, respectively $s \to^* t$. We say that a reduction $s \twoheadrightarrow t$ is *maximal* if there does not exist a term t' such that $t \to t'$. In other words, in this case t is a *normal form*.

▶ **Example 2.5.** Consider the rewrite rule $a \to g(a)$ from the introduction. The following is a strongly convergent, maximal reduction with normal form g^ω:

$$a \to g(a) \to g(g(a)) \to \cdots \to g^n(a) \to \cdots g^\omega\,.$$

Replacing the final term of the reduction by a breaks strong convergence, as the sequence $(g^n(a))_{n<\omega}$ does not converge to a; it only converges to g^ω.

Consider now the rule $c \to c$. We can construct the following reduction of length ω:

$$c \to c \to \cdots \to c \to \cdots c\,.$$

Although a sequence in which all terms are equal to c obviously converges to c, this reduction is not strongly convergent, as the depth of the contracted redexes does not tend to infinity along the reduction.

Compression. Having defined iTRSs and strongly convergent reductions we can now state the classical compression property for left-linear systems [8, 7]:

▶ **Theorem 2.6.** *Let $\mathcal{R}$ be a left-linear iTRS. For each $s \twoheadrightarrow t$ there exists a reduction $s \twoheadrightarrow^{\leq\omega} t$.*

Thus, for every reduction in a left-linear iTRS we can find a reduction with the same initial and final term that is of length at most ω. In the case of *orthogonal* iTRSs the above can be strengthened [8]: a reduction $s \twoheadrightarrow^{\leq\omega} t$ exists which is Lévy equivalent to $s \twoheadrightarrow t$.

3 Generalised Compression

Assuming compression should hold equally for all reductions within a rewrite system, the only obvious generalisation of the compression property is the one from [12]:

▶ **Definition 3.1.** Let α be a countable ordinal, an iTRS $\mathcal{R}$ satisfies the *α-compression property* iff it holds for each $s \twoheadrightarrow t$ that there exists a reduction $s \twoheadrightarrow^{\leq\alpha}$ t.

Since all strongly convergent reductions have countable length [8, 7], only countable ordinals make sense in the definition; the property holds trivially for any uncountable ordinal. By Theorem 2.6, we have that ω-compression holds for every left-linear iTRS.

▶ Remark. The α-compression property is related to the notion of α-closedness from [5]. The difference between the notions is two-fold: First, α-compression allows for both finite and infinite terms as the starting terms of reductions, while α-closedness only allows for finite terms as starting terms. Second, while we consider strongly convergent reductions, α-closedness is concerned with the more general, but less well-behaved, Cauchy convergent or weakly convergent reductions (which dispose of the depth requirement that is part of Definition 2.4 [5, 8]).

Based on the known counterexamples to compression from the literature, one may conjecture that for every iTRS it is possible to find an ordinal α such that α-compression holds: With regard to the standard counterexample from the introduction, [5] states without proof that for *finite* starting terms all reductions are compressible to length at most $\omega + \omega$. A similar property holds for $\lambda\beta\eta$-calculus — the prototypical higher-order system not satisfying the compression property. As can be inferred from [12, Lemma 5], all reductions in $\lambda\beta\eta$-calculus also compress to reductions of length at most $\omega + \omega$.

As we will show below, the above conjecture is false: We uncover two systems that do not satisfy α-compression for any countable α. The first system, discussed in Section 3.1, is — remarkably enough — the one from the standard counterexample. The second system, discussed in Section 3.2, refutes the above conjecture in even stronger ways: We exhibit terms that have unique strongly convergent reductions starting from them which are *incompressible*:

▶ **Definition 3.2.** Let $\mathcal{R}$ be an iTRS, a reduction $s \twoheadrightarrow^{\alpha} t$ is *incompressible* if all reductions $s \twoheadrightarrow t$ are of length at least α.

Considering incompressible reductions mitigates a possible point of critique regarding the proof in Section 3.1: We only show that the considered reductions are at least of a certain length, we do not show that they cannot be compressed precisely up to that length.

3.1 Compression in the Standard Counterexample

Consider the iTRS with as its sole rule $f(x, x) \to c$, the non-left-linear rule from the standard counterexample. We have the following:

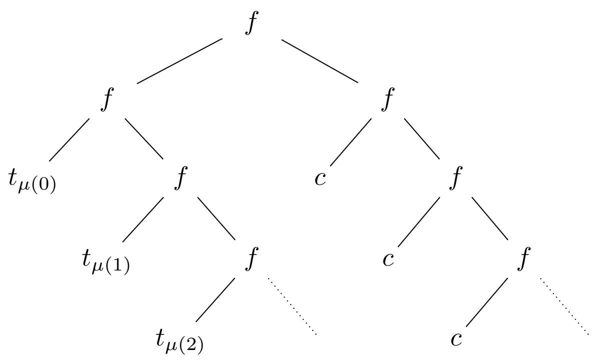


Figure 1 A limit term from the proof of Lemma 3.3.

▶ **Lemma 3.3.** *For every countable ordinal α, there is a term t_α such that $t_\alpha \twoheadrightarrow c$ exists and is of length at least $\alpha + 1$ and such that no $t_\alpha \twoheadrightarrow c$ exists of length less than $\alpha + 1$.*

Proof. We prove the lemma by transfinite induction over the ordinal α. In case $\alpha = 0$, define $t_\alpha = f(c, c)$. As we have $f(c, c) \to c$, we are done.

In case $\alpha = \beta + 1$, we have by the induction hypothesis that a strongly convergent reduction exists from t_β to c. Set $t_\alpha = f(t_\beta, c)$. The reduction $t_\alpha \twoheadrightarrow^{\geq \beta+1} f(c, c) \to c$ is obviously strongly convergent, as $t_\beta \twoheadrightarrow c$ is. Moreover, as the redex at the root is only created after at least $\beta + 1$ steps — for $t_\beta \twoheadrightarrow c$ consists of at least $\beta + 1$ steps — we have that $t_\alpha \twoheadrightarrow c$ is of length at least $\alpha + 1 = \beta + 2$.

In case α is a limit ordinal, choose any bijection μ from $\mathbb{N}$ to α and define $t_\alpha = f(\phi_\mu(0), \psi)$ (see also Figure 1), where:

$$\begin{aligned} \phi_\mu(n) &= f(t_{\mu(n)}, \phi_\mu(n + 1)) \\ \psi &= f(c, \psi) \end{aligned}$$

By the induction hypothesis we have for every $n \in \mathbb{N}$ that a strongly convergent reduction of length at least $\mu(n) + 1$ starts from $t_{\mu(n)}$. Reduce each $t_{\mu(n)}$ to c, possibly interleaving the reductions in the different subterms. Doing so, we obtain $f(\psi, \psi)$, which we can further reduce to c. The reduction to $f(\psi, \psi)$ is strongly convergent, for suppose not, then an infinite number of reduction steps occurs at a certain fixed depth [7, Exercise 12.3.6]. By the pigeonhole principle for infinite sets and since the terms $t_{\mu(n)}$ occur at increasingly greater depths, this means there exists an m such that an infinite number of steps occurs in $t_{\mu(m)}$ at a fixed position. Hence, the reduction from $t_{\mu(m)}$ is not strongly convergent, contradiction. As the reductions from the different $t_{\mu(n)}$ are independent and as a reduction of length at least $\beta + 1$ occurs for every ordinal $\beta < \alpha$, the constructed reduction has length at least $\alpha + 1$. No reduction of length $\gamma < \alpha$ exists, otherwise for infinitely many $t_{\mu(n)}$ with reductions of length at least $\beta + 1 > \gamma$ there also exist reductions of length at most γ, contradicting the induction hypothesis. ◀

Remark that the rules $a \to g(a)$ and $b \to g(b)$ from the standard counterexample do not affect the above result, as a and b do not occur in the defined terms. Hence, the lemma also holds for the iTRS from the standard counterexample.

By the above, we immediately have:

▶ **Theorem 3.4.** *There exists an iTRS such that the α-compression property fails to hold for every countable ordinal α.*

3.2 Unique Incompressible Reductions

Although the result from the previous section establishes that compression, even in a generalised form, cannot be carried over to arbitrary iTRSs, the result is not completely satisfactory: We only uncovered reductions that are of *at least* a certain length; we did not show that these reductions cannot be compression at all (up to the provided lower bounds on their lengths). To rectify this situation, we study a second system with a non-left-linear rule in the current section. For this second system, we will define for every countable ordinal α terms t_{ν_α} with starting from them unique maximal and incompressible reductions of length *at least* α. As a corollary we will obtain that each t_{ν_α} also has a unique incompressible reduction of length *precisely* α starting from it.

Before we give the actual proofs, we first describe our second system and give some examples.

Signature. The signature Σ of our second system consists of five symbols:

$$\Sigma = \{f, f', h, h', k\}\,,$$

where f and f' are ternary and such that all other symbols are unary. Of these symbols, f is the most important and is the root symbol of our unique non-left-linear rule. The symbol h will be inserted above the redexes we are going to contract to ensure that the redexes occur at sufficient depth to guarantee strong convergence. Moreover, the symbols f' and h' will allow us to construct terms that are already of the required 'shape', but in which no redexes occur (due to the presence of the primes). Finally, the function symbol k will help us to convert 'primed' into 'unprimed' terms.

Rewrite Rules. Our system has four rewrite rules: One non-left-linear rule and three rules to convert 'primed' into 'unprimed' terms. The non-left-linear rule is as follows:

$$f(x, x, y) \to h(k(y)) \tag{1}$$

That is, we match the first two arguments of f.

Contracting an f-redex introduces an h to ensure that the term substituted for y will eventually — after contracting of a number of k-redexes — occur at the same depth as before contraction of the f-redex.

As already mentioned, the rules associated with k convert 'primed' into 'unprimed' terms. The rules are as follows:

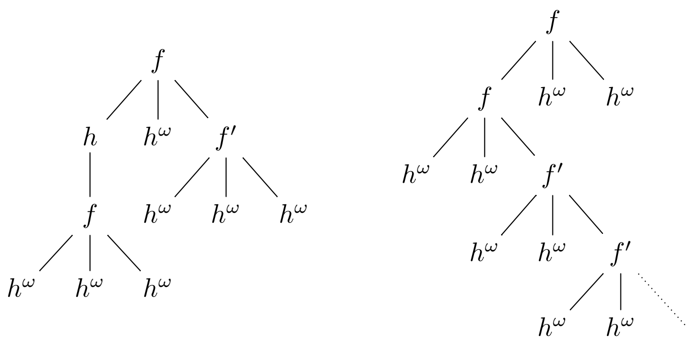
$$k(f'(x, y, z)) \to f(k(x), y, z) \tag{2}$$

$$k(h'(x)) \to h(k(x)) \tag{3}$$

$$k(h(x)) \to h(x) \tag{4}$$

The first two rules remove a prime and then recurse. The third rule ensures that the conversion terminates once we encounter a non-primed h. Although a similar terminating rule could be introduced for f, i.e. $k(f(x, y, z)) \to f(x, y, z)$, the terms considered below are such that this rule is never needed and, hence, we prefer omit the rule.

▶ Remark. Observe that the first two rules associated with k move the k to a lower position and that the last rule removes the k all together. Hence, if we would instead use $f(x, x, y) \to k(y)$ as our f-rule, it not longer holds that the (converted) term substituted for y eventually occurs at the same depth d as before application of the f-rule; it will occur at depth $d - 1$ instead. The equal depth is, however, needed to ensure that our reductions are strongly convergent. For this reason the h occurs at the root of the right-hand side of our f-rule.



(a) Example 3.5. **(b)** Example 3.6.

Figure 2 The initial terms from Examples 3.5 and 3.6.

Two Examples. Before proving the central result of this section, let us consider two concrete incompressible reductions that make use of the above rules.

▶ **Example 3.5.** Consider the following term (also depicted in Figure 2(a)):

$$s = f(h(f(h^\omega, h^\omega, h^\omega)), h^\omega, f'(h^\omega, h^\omega, h^\omega))\,.$$

Underlining contracted redexes, we have precisely one maximal reduction starting from s (consisting of seven steps):

$$\begin{aligned} f(h(\underline{f(h^\omega, h^\omega, h^\omega)}), h^\omega, f'(h^\omega, h^\omega, h^\omega)) &\to f(h(h(\underline{k(h^\omega)})), h^\omega, f'(h^\omega, h^\omega, h^\omega)) \\ &\to \underline{f(h^\omega, h^\omega, f'(h^\omega, h^\omega, h^\omega))} \to h(\underline{k(f'(h^\omega, h^\omega, h^\omega))}) \\ &\to h(f(\underline{k(h^\omega)}, h^\omega, h^\omega)) \to h(\underline{f(h^\omega, h^\omega, h^\omega)}) \to h(h(\underline{k(h^\omega)})) \to h^\omega\,. \end{aligned}$$

This reduction cannot be compressed to a reduction of less than seven steps: For each step the redex contracted in that step is created in the step immediately preceding it.

With regard to the above example, note that, if we fix the ordinal α to be 3, we obtain a bijection ν_α^{-1} between α and the depths of the α $(= 3)$ f-steps in the reduction: $\nu_\alpha^{-1}(0) = 2$, $\nu_\alpha^{-1}(1) = 0$, and $\nu_\alpha^{-1}(2) = 1$; the first f-step occurs at depth 2, the second at depth 0, and the third at depth 1. Bijections like ν_α^{-1} will be central in the construction of reductions such as the above. As the inverse of the bijection will be more important below, we choose to write ν_α^{-1} here instead of ν_α.

▶ **Example 3.6.** Consider the following system of recursive equations, with the solution of s depicted in Figure 2(b):

$$\begin{aligned} s &= f(f(h^\omega, h^\omega, t), h^\omega, h^\omega) \\ t &= f'(h^\omega, h^\omega, t) \end{aligned}$$

As in the previous example, we have exactly one maximal reduction starting from s (in this case one of length $\omega + 2$):

$$\begin{aligned} f(\underline{f(h^\omega, h^\omega, t)}, h^\omega, h^\omega) &\to f(h(\underline{k(t)}), h^\omega, h^\omega) \to f(h(f(\underline{k(h^\omega)}, h^\omega, t)), h^\omega, h^\omega) \\ &\to f(h(\underline{f(h^\omega, h^\omega, t)}), h^\omega, h^\omega) \to f(h^2(\underline{k(t)}), h^\omega, h^\omega) \to \cdots \\ &\to f(h^n(\underline{k(t)}), h^\omega, h^\omega) \to \cdots \underline{f(h^\omega, h^\omega, h^\omega)} \to h(\underline{k(h^\omega)}) \to h^\omega\,. \end{aligned}$$

Like the reduction from the previous example, the above reduction cannot be compressed, as each contracted redex is created in the step immediately preceding it.

As before, we have with regard to the above example that a bijection ν_α^{-1} exists from an ordinal α (in this case $\omega + 1$) to the depths of the α f-steps that occur along the reduction: $\nu_\alpha^{-1}(i) = i + 1$ for all $i < \omega$ and $\nu_\alpha^{-1}(\omega) = 0$.

Initial Term Construction. Following the above examples, our incompressible reductions will be defined by specifying initial terms and showing that each of these initial terms allows for exactly one maximal reduction (which cannot be compressed).

To define the initial terms, we first introduce a function $\rho : \mathcal{T}er(\Sigma, V) \to \mathcal{T}er(\Sigma, V)$ which maps 'unprimed' terms to 'primed' ones:

$$\rho(f(s_1, s_2, s_3)) = f'(\rho(s_1), s_2, s_3)$$

$$\rho(h(s)) = \begin{cases} h'(\rho(s)) & \text{if } s \neq h^\omega \\ h^\omega & \text{if } s = h^\omega \end{cases}$$

for all $s_1, s_2, s_3, s \in \mathcal{T}er(\Sigma, V)$.

The rewrite rules for k 'cancel out' ρ:

▶ **Lemma 3.7.** *Let $t \in \mathcal{T}er(\Sigma, V)$. It holds that $k(\rho(t)) \twoheadrightarrow^{\leq\omega} t$. Moreover, if $t|_{1^n} = h^\omega$ for some $n \in \mathbb{N}$, then $k(\rho(t)) \twoheadrightarrow^{\leq n} t$.*

Proof. By definition of ρ, we have that exactly one maximal strongly convergent reduction can be defined starting from $\rho(t)$ and employing only k-rules. Also by definition of ρ, the reduction is finite in case $t|_{1^n} = h^\omega$ for some $n \in \mathbb{N}$. That the final term of the reduction is t follows by structural induction over the positions of t, observing that $k(\rho(f(s_1, s_2, s_3))) \to f(k(\rho(s_1)), s_2, s_3)$ and that, moreover, $k(\rho(h(s))) \to h(k(\rho(s)))$ in case $s \neq h^\omega$, and $k(\rho(h(s))) \to h^\omega$ in case $s = h^\omega$. ◀

We can now define the initial terms of our incompressible reductions, where it is strongly suggested that the reader satisfies him- or herself of the fact that the initial terms from Examples 3.5 and 3.6 can be constructed by employing this definition.

▶ **Definition 3.8.** Let α be a countable ordinal and $\nu_\alpha : D \to \alpha$ a bijection with $D \subseteq \mathbb{N}$. The term t_{ν_α} is defined by $\tau_{\nu_\alpha}(0, 0, \alpha)$ where $\tau_{\nu_\alpha} : \mathbb{N} \times (\alpha + 1)^2 \to \mathcal{T}er(\Sigma, V)$ is such that:

$$\tau_{\nu_\alpha}(d, \delta, \gamma) = \begin{cases} f(\tau_{\nu_\alpha}(d+1, \delta, \nu_\alpha(d)), \\ \qquad h^\omega, \rho \circ \tau_{\nu_\alpha}(d+1, \nu_\alpha(d)+1, \gamma)) & \text{if } d \in D \text{ and } \nu_\alpha(d) \in [\delta, \gamma) \\ h(\tau_{\nu_\alpha}(d+1, \delta, \gamma)) & \text{otherwise} \end{cases}$$

With regard to the f-steps from the reductions we will be constructing, ν_α^{-1} indicates the depth at which each of these steps occurs. The first parameter of τ_{ν_α} specifies the depth of the subterm of t_{ν_α} we are currently constructing; the second and third parameter indicate the range of f-steps that will occur within this subterm.

The first clause of τ_{ν_α} defines a subterm at depth d with an f at the root in case an f-step should occur at depth d and is within the current range (i.e. $\nu_\alpha(d) \in [\delta, \gamma)$). Moreover, the range is split over the two occurrences of τ_{ν_α}: All earlier f-steps at greater depths occur in the first argument of f and all later f-steps at greater depths occur in the third argument. Note that the second argument of f is always h^ω and, hence, f is not the root of a redex unless the first argument is also equal to h^ω. Moreover, note that no redexes occur in the third argument of f due to the occurrence of ρ.

The second clause of τ_{ν_α} defines a subterm at depth d with h at the root in case no f-step from within the current (possibly empty) range occurs at d.

Every subterm $\tau_{\nu_\alpha}(d, \delta, \gamma)$, in particular $t_{\nu_\alpha} = \tau_{\nu_\alpha}(0, 0, \alpha)$, satisfies the following property.

▶ **Lemma 3.9.** *Suppose $\tau_{\nu_\alpha}(d, \delta, \gamma)$ is such that for all $d' \geq d$ it holds that either $d' \notin D$ or $\nu_\alpha(d') \notin [\delta, \gamma)$, then $\tau_{\nu_\alpha}(d, \delta, \gamma) = h^\omega$.*

Proof. By induction on $d' \geq d$, where we have for every $d' \geq d$ that the second clause from the definition of τ_{ν_α} applies. ◀

We also have:

▶ **Lemma 3.10.** *For every $\tau_{\nu_\alpha}(0, \delta, \gamma)$ with $\nu_\alpha(d) \in [\delta, \gamma)$ for some $d \in D$ there exists an $n \in \mathbb{N}$ such that $\tau_{\nu_\alpha}(0, \delta, \gamma)|_{1^n} = f(h^\omega, h^\omega, t)$ for some term t.*

Proof. Suppose not, then either $\tau_{\nu_\alpha}(0, \delta, \gamma) = h^\omega$ or eventually always the first clause from the definition of τ_{ν_α} applies. In the first case, no d exists such that $\nu_\alpha(d) \in [\delta, \gamma)$, contradiction. In the second case, there exists an infinite chain of depths $d_1 < d_2 < \ldots < d_n < \ldots$ with $[\delta, \nu_\alpha(d_1)) \supsetneq [\delta, \nu_\alpha(d_2)) \supsetneq \ldots \supsetneq [\delta, \nu_\alpha(d_n)) \supsetneq \ldots$ Hence, $\nu_\alpha(d_1) > \nu_\alpha(d_2) > \ldots > \nu_\alpha(d_n) > \ldots$ However, this is an infinite descending chain of ordinals, which cannot exist, again a contradiction. ◀

Incompressible Reductions. Having introduced all necessary ingredients, we can now prove the central theorem of this section.

▶ **Theorem 3.11.** *For every ν_α, the term t_{ν_α} has an unique maximal strongly convergent reduction starting from it which is incompressible and of length at least length α.*

Proof. We prove by transfinite induction that $t_{\nu_\alpha} = \tau_{\nu_\alpha}(0, 0, \alpha)$ reduces to $\tau_{\nu_\alpha}(0, \kappa, \alpha)$ with $\kappa \leq \alpha$ in at least κ rewrite steps. In case $\kappa = 0$, the result is immediate.

In case $\kappa = \lambda + 1$, we have by the induction hypothesis that t_{ν_α} reduces to $\tau_{\nu_\alpha}(0, \lambda, \alpha)$ in at least λ steps. Hence, it suffices to show that $\tau_{\nu_\alpha}(0, \lambda, \alpha)$ reduces to $\tau_{\nu_\alpha}(0, \kappa, \alpha)$ in at least one step. As $\lambda < \alpha$, we have $\nu_\alpha(d) \in [\lambda, \alpha)$ for some $d \in D$ and, hence, by Lemma 3.10, a redex occurs at a position 1^n for some $n \in \mathbb{N}$. In fact, by definition of τ_{ν_α}, $n = \nu_\alpha^{-1}(\lambda)$. There are now two cases to consider depending on the value of $\nu_\alpha^{-1}(\kappa)$: We either have $\nu_\alpha^{-1}(\kappa) < n$ or $\nu_\alpha^{-1}(\kappa) > n$ (equality is impossible, as ν_α is a bijection). We consider each of these cases in turn.

- In case $\nu_\alpha^{-1}(\kappa) < n$, the redex at position 1^n is defined by $\tau_{\nu_\alpha}(n, \lambda, \kappa)$. Hence, since $\kappa = \lambda + 1$, it follows by definition of τ_{ν_α} and Lemma 3.9 that $\tau_{\nu_\alpha}(n, \lambda, \kappa) = f(h^\omega, h^\omega, h^\omega)$. We have:

  $$f(h^\omega, h^\omega, h^\omega) \to h(k(h^\omega)) \to h^\omega .$$

  Thus, $\tau_{\nu_\alpha}(n, \lambda, \kappa)$ reduces to $\tau_{\nu_\alpha}(n, \kappa, \kappa) = h^\omega$ in two steps and the result follows by observing that $\tau_{\nu_\alpha}(0, \lambda, \alpha)$ and $\tau_{\nu_\alpha}(0, \kappa, \alpha)$ are identical except for the subterm at 1^n.
- In case $\nu_\alpha^{-1}(\kappa) > n$, the redex at position 1^n is defined by $\tau_{\nu_\alpha}(n, \lambda, \iota)$ for some $\iota > \kappa$. Hence, the subterm at position 1^n is of the form $f(h^\omega, h^\omega, \rho \circ \tau_{\nu_\alpha}(n + 1, \kappa, \iota))$. We have

  $$f(h^\omega, h^\omega, \rho \circ \tau_{\nu_\alpha}(n + 1, \kappa, \iota) \to h(k(\rho \circ \tau_{\nu_\alpha}(n + 1, \kappa, \iota)))$$

  and by Lemma 3.7, this further reduces to $h(\tau_{\nu_\alpha}(n+1, \kappa, \iota)) = \tau_{\nu_\alpha}(n, \kappa, \iota)$. By these facts and the observation that $\tau_{\nu_\alpha}(0, \lambda, \alpha)$ and $\tau_{\nu_\alpha}(0, \kappa, \alpha)$ are identical except for the subterm at position 1^n, the result follows.

In case κ is a limit ordinal, observe that whenever λ approaches κ from below, we have that the redexes occur at increasingly greater depths. In case of the f-redexes, this follows as the position at which the λth f-redex occurs is equal to $1^{\nu_\alpha^{-1}(\lambda)}$ and as ν_α is a bijection. In case of the k-redexes, we have that only finitely many of these redexes are contracted after each f-step. Moreover, all these redexes occur at depths greater than $\nu_\alpha^{-1}(\lambda)$. Hence, as the redexes occur at increasingly greater depths, it follows by the induction hypothesis that we have a strongly convergent reduction which passes through each $\tau_{\nu_\alpha}(0, \lambda, \alpha)$ with $\lambda < \kappa$. As $\tau_{\nu_\alpha}(0, \lambda, \alpha)$ and $\tau_{\nu_\alpha}(0, \kappa, \alpha)$ are identical except for some subterm at a position 1^n, where n eventually increases as λ increases (as ν_α is a bijection), $\tau_{\nu_\alpha}(0, \kappa, \alpha)$ is the final term of the strongly convergent reduction, concluding this case.

Our theorem follows once we observe that each redex contracted in the constructed reduction (of length at least α) is created in the step immediately preceding it. ◀

We now immediately have:

▶ **Corollary 3.12.** *For every countable ordinal α, there exists a strongly convergent reduction of length* at least *α that is incompressible.*

In fact, as each prefix of each of the above reductions is also incompressible, we obtain the following by the same reasoning regarding redex creation:

▶ **Corollary 3.13.** *For every countable ordinal α, there exists a strongly convergent reduction of length* precisely *α that is incompressible.*

4 Standardisation

Having established that compression does not generalise to systems with non-left-linear rules, we will next reinterpret compression in *left-linear* systems as a degenerate case of standardisation. To be able to do this, we first need to define standard reductions.

Starting from finite rewriting, we could define a standard reduction as one that contracts redexes in leftmost-outermost order [14, Section 8.5], where a position p_1 is said to occur to the left of a position p_2 if $p_1 = q \cdot n_1 \cdot p_1'$ and $p_2 = q \cdot n_2 \cdot p_2'$ with $n_1 < n_2$ for some q (note that $p_1 \parallel p_2$). However, this notion of standardisation is ill-suited to our purposes, as infinite terms exist that do not have a leftmost redex.

▶ **Example 4.1.** Consider the term which is the unique solution of the equation $s = f(s, a)$ and suppose we have at our disposal the rewrite rule $a \to b$. The term s does not have a leftmost-outermost redex: For each $n \in \mathbb{N}$, an outermost redex occurs at position $1^n \cdot 2$. However, no redex is leftmost, as also for each $n \in \mathbb{N}$ the redex at position $1^{n+1} \cdot 2$ occurs to the left of the redex at position $1^n \cdot 2$.

Dropping the requirement that the order in standard reductions should be leftmost, we can alternatively consider *parallel standard reductions* [14, Definition 8.5.6], which allow for more freedom with regard to redexes that occur at parallel positions. The definition is as follows, where we make explicit the case distinction that implicitly occurs in [14, Definition 8.5.6]:

▶ **Definition 4.2.** Let $t_0 \twoheadrightarrow t_\alpha$ with $(p_\beta, l_\beta \to r_\beta)_{\beta<\alpha}$ the sequence of rewrite steps of $t_0 \twoheadrightarrow t_\alpha$. The reduction $t_0 \twoheadrightarrow t_\alpha$ is *parallel standard* iff for every $\beta < \alpha$ either:

- $p_\beta \parallel p_\kappa$ or $p_\beta \leq p_\kappa$ for all $\beta < \kappa < \alpha$, or
- $p_\beta = p_\kappa \cdot p_\beta'$ with $p_\beta' \in \{q \in \mathcal{P}\mathrm{os}(l_\kappa) \mid \mathrm{root}(l_\kappa|_q) \in \Sigma\}$ and $\kappa = \min\{\gamma \in (\beta, \alpha) \mid p_\beta > p_\gamma\}$.

Hence, either (a) every step after the βth step should occur parallel to or below p_β, or (b) the position p_β should occur either in the redex pattern of the first redex that occurs at a position p_κ above p_β (i.e. $p_\beta = p_\kappa \cdot q$ with q a non-variable position of l_κ). Thus, in the second case, contracting the redex in the βth step helps to create the redex pattern of the redex contracted in the κth step.

As in the finite case, parallel standard reductions are not necessarily unique: Given the rule $a \to h(a)$, we have that both $f(\underline{a}, a) \to f(h(a), \underline{a}) \to f(h(a), h(a))$ and $f(a, \underline{a}) \to f(\underline{a}, h(a)) \to f(h(a), h(a))$ are parallel standard reductions from $f(a, a)$ to $f(h(a), h(a))$. We will further discuss the issue of uniqueness in Section 4.2.

▶ Remark. Compressed reductions do not need to be parallel standard: Consider the rules $a \to h(a)$ and $f(x) \to g(x)$. The reduction $f(a) \to f(h(a)) \to g(h(a))$ is compressed, as it has length $\leq \omega$. However, it is not parallel standard, as the a-redex that is being contracted occurs below the contracted f-redex, while it is not part of the redex pattern of the f-redex.

Although Definition 4.2 alleviates the problem with leftmost redexes as exhibited in Example 4.1, it does allow for standard reductions of length greater than ω.

▶ **Example 4.3.** Consider the term $f(a, a)$ and the rewrite rule $a \to h(a)$ from above. We have the following reduction of length $\omega + \omega$:

$$\begin{aligned} &f(a,a) \to f(h(a),a) \to \cdots \to f(h^n(a),a) \to f(h^{n+1}(a),a) \to \cdots f(h^\omega,a) \\ &\qquad \to f(h^\omega,h(a)) \to \cdots \to f(h^\omega,h^n(a)) \to f(h^\omega,h^{n+1}(a)) \to \cdots f(h^\omega,h^\omega)\,. \end{aligned}$$

The reduction is parallel standard, as the first argument of f is parallel to its second argument.

The situation in the above example can be mitigated by strengthening Definition 4.2. One possibility with regard to the first clause of the definition is to not only consider $p_\beta \parallel p_\kappa$ and $p_\beta \leq p_\kappa$, but additionally $|p_\beta| \leq |p_\kappa|$, which when combined reduces to simply $|p_\beta| \leq |p_\kappa|$. With this restriction the reduction from the above example is no longer valid: contraction of a-redexes in the second argument is postponed even though we are contracting redexes at greater depths in the first argument.

We prefer not to introduce the above strengthening as part of the definition of parallel standard reductions as we wish to stay as close as possible to the definition from finite rewriting. Our standardisation procedure does, however, implicitly use the strengthening.

4.1 Parallel Standardisation

To show that reductions in left-linear systems can be transformed into parallel standard form, we first introduce a variant of parallel standard reduction which is limited to a certain depth.

▶ **Definition 4.4.** Let $t_0 \twoheadrightarrow t_\alpha$ with $(p_\beta, l_\beta \to r_\beta)_{\beta<\alpha}$ the sequence of rewrite steps of $t_0 \twoheadrightarrow t_\alpha$. The reduction $t_0 \twoheadrightarrow t_\alpha$ is *parallel standard up to depth* $d \in \mathbb{N}$ iff for every $\beta < \lambda$ with $\lambda = \max\{\gamma \mid |p_\gamma| < d\}$ either:

- $|p_\beta| \leq |p_\kappa|$ for all $\beta < \kappa \leq \lambda$, or
- $p_\beta = p_\kappa \cdot p'_\beta$ with $p'_\beta \in \{q \in \mathcal{P}\mathrm{os}(l_\kappa) \mid \mathrm{root}(l_\kappa|_q) \in \Sigma\}$ and $\kappa = \min\{\gamma \in (\beta, \lambda] \mid p_\beta > p_\gamma\}$.

Thus, a reduction is parallel standard up to depth d if (a) the reduction is parallel standard up to and including the last step in $t_0 \twoheadrightarrow t_\alpha$ contracting a redex at depth less than d and (b) it incorporates the depth requirement on parallel redexes as mentioned immediately below Example 4.3.

Our standardisation procedure works on a depth-by-depth basis and per depth follows the inversion procedure from [14, Section 8.5.3]. Effectively, this means that per depth, standardisation will be achieved by permuting redexes that are not parallel standard.

Permutation. Write $s \to_{\parallel} t$ for a reduction contracting (an infinite number of) parallel redexes; such a reduction can always be transformed into a strongly convergent reduction, e.g. by contracting the parallel redexes in order of least depth. We have the following lemma, assuming left-linearity.

▶ **Lemma 4.5** (Step Permutation). *Let $d \in \mathbb{N}$. If $s \to_{\parallel} t \to s'$ is such that all steps in $s \to_{\parallel} t$ occur at depth $> d$ and such that $t \to s'$ occurs at a position p with $|p| \geq d$, then there exists a reduction $s \to^* t' \to_{\parallel} s'$ which is parallel standard up to depth $d+1$ such that all steps occur at depth $\geq d$ and such that the final step of $s \to^* t'$ occurs at position p.*

Proof. As left-hand sides of rewrite rules are finite, only finitely many steps from $s \to_{\parallel} t$ are required to create the redex at position p contracted in $t \to s'$ and to guarantee parallel standardness; the *required* steps are those at positions q such that either $q \leq p$ or $q = p \cdot q'$ with $q'(\neq \epsilon)$ occurring in the redex pattern of the redex contracted in $t \to s'$. Define $s \to^* t''$ by first contracting all the required redexes in order of least depth and next contracting the redex at position p, which exists by contraction of the required redexes and left-linearity.

Project the remaining redexes from $s \to_{\parallel} t$ over $s \to^* t''$. By left-linearity and since the original set of redexes is parallel, this projection yields a set of redexes all of which are parallel and occur at depth $> d$, possibly, with exception of a unique redex occurring at position p with $|p| = d$ (in which case $t \to s'$ is collapsing). In this latter case define t' to be the result of contracting the parallel redex at position p in t'', otherwise define t' to be t''. Contracting the remaining parallel redexes yields s', as a projection argument similar to the Strip Lemma for orthogonal iTRSs [8, 7] shows.

It is easily seen that all steps in $s \to^* t' \to_{\parallel} s'$ occur at depth $\geq d$ and that the final step of $s \to^* t'$ occurs at position p (and depth d in case $|p| = d$). Moreover, by construction it also immediately follows that the reduction is parallel standard up to depth $d+1$. ◀

Write $s \to_{\parallel}^* t$ for a finite sequence of parallel steps, we can now use the previous result to permute steps in finite reductions, again assuming left-linearity.

▶ **Lemma 4.6** (Reduction Permutation). *Let $n \geq 1$ and $d \in \mathbb{N}$. If $s_0 \to^* s_n$ is such that all steps in $s_0 \to^* s_{n-1}$ occur at depth $> d$ and such that $s_{n-1} \to s_n$ contracts a redex at a position p with $|p| = d$, then there exists a reduction $s_0 \to^* t \to_{\parallel}^* s_n$ which is parallel standard up to depth $d+1$ such that all steps occur at depth $\geq d$ and such that the last step of $s_0 \to^* t$ is the last step that occurs at position p and depth d.*

Proof. The proof is by induction on the number of steps in $s_0 \to^* s_n$, where in addition we show that the second clause of Definition 4.4 applies to every step at depth $> d$ in $s_0 \to^* t$. In case $n = 1$, the reduction consists of a single step and the result is immediate.

In case $n > 1$, write $s_0 \to s_1 \to^* s_n$. By the induction hypothesis there exists a reduction $s_1 \to^* t' \to_{\parallel}^* s_n$ which is parallel standard up to $d+1$ and which satisfies the required properties. We construct $s_0 \to^* t$ by permuting a set of parallel redexes such that at any point during the permutation we have a reduction of the form $s_0 \to^* t_1 \to_{\parallel} t_2 \to^* t'$. Initially, define $s_0 \to^* t_1$ to be empty, $t_1 \to_{\parallel} t_2$ equal to $s_0 \to s_1$, and $t_2 \to^* t'$ equal to $s_1 \to^* t'$. For each permutation step, write $t_2 \to^* t'$ as $t_2 \to t_2' \to^* t'$ and apply Lemma 4.5 to $t_1 \to_{\parallel} t_2$ and $t_2 \to t_2'$ to obtain $t_1 \to^* t_1' \to_{\parallel} t_2'$. In the following permutation step assume that $s_0 \to^* t_1$ is equal to $s_0 \to^* t_1'$, that $t_1 \to_{\parallel} t_2$ is equal to $t_1' \to_{\parallel} t_2'$, and that $t_2 \to^* t'$ is equal to $t_2' \to^* t'$. Continue until $t_2 \to^* t'$ is empty and then define t to be t_1 and $t \to_{\parallel}^* s_n$ to be $t_1 \to_{\parallel} t' \to_{\parallel}^* s_n$.

By the permutation procedure and Lemma 4.5, we have that all steps in $s_0 \to^* t \to_{\parallel}^* s_n$ occur at depth $\geq d$ and that the last step of $s_0 \to^* t$ is the last step that occurs at position p and depth d. Hence, this leaves to show that $s_0 \to^* t$ is parallel standard up to depth $d+1$

such that the second clause of Definition 4.4 applies to any step at depth $> d$: Since all steps occur at depth $\geq d$, the first clause of Definition 4.4 immediately applies to any step at depth d. For any other step it follows by the construction in the proof of Lemma 4.5 and the permutation procedure that any such step either (a) occurs in the redex pattern of a step following it or (b) occurs above it. In the first case, the second clause holds immediately; in the second case, observe that there must be some later step in whose redex pattern the step under consideration occurs (thus implying the second clause), otherwise, tracing usage of redex patterns in contracted redexes, it follows that the second clause does not apply to some of the reduction steps at depth $> d$ in $s_1 \to^* t'$, contradicting the induction hypothesis. ◀

Needed Rewrite Steps. To work on a depth by depth basis in our standardisation theorem, we require a way to establish which redexes are needed for the creation of a redex at a certain depth. To this end we extend from finite rewriting [4] the notion of origins (roughly the inverse of descendants) and we define the related notion of needed steps.

▶ **Definition 4.7.** Let $s \to t$ be adorned with $(p, l \to r)$. If $q \in \mathcal{P}os(t)$, then the set of *origins of q across $s \to t$*, denoted $(s \to t) \nwarrow q$, is the set of positions of s defined as follows:

- $\{q\}$ if $q \parallel p$ or $q < p$,
- $\{p \cdot q' \mid \mathrm{root}(l|_{q'}) \in \Sigma\}$ if $q = p \cdot p'$ with $\mathrm{root}(r|_{p'}) \in \Sigma$, and
- $\{p \cdot q' \cdot p'' \mid l|_{q'} = r|_{p'}\} \cup \{p \cdot q' \mid l|_{q'} \in \Sigma \text{ and } r|_\epsilon \in V\}$ if $q = p \cdot p' \cdot p''$ with $r|_{p'} \in V$.

If Q is a finite set of positions, then $(s \to t) \nwarrow Q = \bigcup_{q \in Q} (s \to t) \nwarrow q$; if $t_0 \to^* t_n$ is a finite reduction, then $(t_0 \to^* t_n) \nwarrow Q = (t_0 \to^* t_{n-1}) \nwarrow ((t_{n-1} \to t_n) \nwarrow Q)$. Moreover, if $t_0 \twoheadrightarrow t_\alpha$, then $(t_0 \twoheadrightarrow t_\alpha) \nwarrow Q = (t_0 \twoheadrightarrow t_\gamma) \nwarrow ((t_\gamma \to^* t_\beta) \nwarrow Q)$ with γ the largest limit ordinal smaller than or equal to β (and $\gamma = 0$ if no such ordinal exists) and β such that each step in $t_\beta \twoheadrightarrow t_\alpha$ occurs at a position p with $|p| > |q|$ for all $q \in Q$.

Remark that the set of origins of a finite set Q is finite, as left-hand sides of rewrite rules are finite. Moreover, if Q is prefix-closed (i.e. $p < q \in Q$ implies $p \in Q$), then the set of origins is also prefix-closed.

▶ **Example 4.8.** Consider the reduction rules $f(x) \to h(x, x)$ and $g(x) \to x$. We have

$$(f(g(a)) \to h(g(a), g(a)) \to h(a, g(a))) \nwarrow \{\epsilon, 1, 2, 21\}$$
$$= (f(g(a)) \to h(g(a), g(a))) \nwarrow \{\epsilon, 1, 11, 2, 21\} = \{\epsilon, 1, 11\}\,.$$

We now define needed steps.

▶ **Definition 4.9.** Let $t_0 \twoheadrightarrow t_\alpha$ and let Q be a finite, prefix-closed subset of positions of t_α. A step $(p_\beta, l_\beta \to r_\beta)$ is *needed for Q* iff $p_\beta \in (t_\beta \twoheadrightarrow t_\alpha) \nwarrow Q$.

Observe by strong convergence and the definition of origins that only finitely many redexes are needed for finite, prefix-closed subsets of positions of t_α. In case of Example 4.8 both the first and second step are needed for $\{\epsilon, 1\}$; only the first step is needed for $\{\epsilon\}$ and $\{\epsilon, 2, 21\}$.

Standardisation. Employing neededness, we can now prove our standardisation theorem for left-linear iTRSs; the argument derives from the compression argument in [10]. It is, however, necessarily more complicated due to added complexity of standardisation over compression.

▶ **Theorem 4.10.** *Let $s \twoheadrightarrow t$ in a left-linear iTRS. There exists a parallel standard reduction of length at most ω from s to t.*

Proof. We prove by induction over the depth $d \in \mathbb{N}$ that there exists a reduction $s \to^* s_d \twoheadrightarrow t$ which is parallel standard up to depth d and such that all steps in $s_d \twoheadrightarrow t$ occur at depth $\geq d$. In case $d = 0$, define $s_0 = s$. The result is immediate in this case.

In case $d > 0$, it follows by the induction hypothesis that there exists a reduction $s \to^* s_{d-1} \twoheadrightarrow t$ which is parallel standard up to depth $d - 1$ such that all steps in $s_{d-1} \twoheadrightarrow t$ occur at depth $\geq d - 1$. There are now two cases to consider, either no redex in the reduction $s_{d-1} \twoheadrightarrow t$ occurs at depth d or there are such redexes. In the first case, $s \to^* s_{d-1}$ is also parallel standard up to depth d and it suffices to define $s_d = s_{d-1}$. In the remainder of the proof we consider the second case.

Consider the first redex in $s_{d-1} \twoheadrightarrow t$ which occurs at depth d and suppose that the step in which the redex is contracted is $s' \to t'$. Let Q be the smallest prefix-closed set of positions in s' which includes the positions in the redex pattern of the redex contracted in $s' \to t'$. By definition of neededness, finitely many steps from $s_{d-1} \twoheadrightarrow s'$ are needed for Q, all of which occur at depth $> d$ by definition of $s_{d-1} \twoheadrightarrow t$. Moreover, by definition of origins, these needed steps (in their original order) together with the redex contracted in $s' \to t'$ form a finite reduction from s_{d-1} to some term t''. By a projection argument, similar to the Strip Lemma for orthogonal iTRSs [8, 7], it follows that $s_{d-1} \to^* t'' \twoheadrightarrow t$, where the steps at depth d are identical to those of $s_{d-1} \twoheadrightarrow t$. Observe now that Lemma 4.6 can be applied to $s_{d-1} \to^* t''$, as all steps needed for Q occur at depth $> d$ and as $s' \to t'$ occurs at depth d. Let $s_{d-1} \to^* s'_{d-1} \twoheadrightarrow t''$ be the result of applying Lemma 4.6. By definition of $s_{d-1} \to^* s'_{d-1} \twoheadrightarrow t''$ and the observation that the second clause of Definition 4.4 applies to all steps in $s_{d-1} \to^* s'_{d-1}$ at depth $> d$ (see the proof of Lemma 4.6), it follows that $s \to^* s_{d-1} \to^* s'_{d-1}$ is parallel standard up to depth $d + 1$ and that the number of redexes at depth d in $s'_{d-1} \twoheadrightarrow t'' \twoheadrightarrow t$ is equal to the number of redexes at depth d in $t' \twoheadrightarrow t$ (i.e. one less than the number of redexes at depth d in $s' \twoheadrightarrow t$). Hence, we can now repeat the argument with $s'_{d-1} \twoheadrightarrow t$ until no redexes are left at depth d, at which point we have obtained a reduction $s \to^* s_d \twoheadrightarrow t$ which is parallel standard up to depth $d + 1$.

As the redexes considered above occur at increasingly greater depths, we have that the constructed reduction is strongly convergent. Moreover, the reduction is of length at most ω, as $s \to^* s_d$ is finite for each $d \in \mathbb{N}$. Finally, the reduction is parallel standard, as it is parallel standard up to every depth $d \in \mathbb{N}$. ◀

As the above proof applies to arbitrary reductions, the following is now immediate.

▶ **Corollary 4.11.** *There exists a standardisation procedure which transforms every reduction of every left-linear iTRS into a parallel standard reduction of length at most ω.*

Hence, standardisation can be used to obtain a compressed reduction. Turning this around and observing that the proof of the compression theorem is also based on permutation of redexes [8, 7], we can also see the compression theorem as a degenerate instance of a standardisation procedure. It is degenerate, as compression is less strict with respect to the order in which redexes are contracted. As such, we have arrived at our reinterpretation of compression.

4.2 Approximating Leftmost-Outermost Standardisation

Although it is not always possible to obtain a reduction which is standard in the sense of the leftmost-outermost order (see Example 4.1), we would still like to approximate this order to ensure that there is a unique standard reduction as in the finite case (see Theorem 8.5.51 and Lemma 8.5.52 in [14]). The natural idea in the context of infinitary rewriting, as also mentioned in [1], is to take into account depth next to the left-to-right order.

▶ **Definition 4.12.** Let $t_0 \twoheadrightarrow t_\alpha$ with $(p_\beta, l_\beta \to r_\beta)_{\beta<\alpha}$ the sequence of rewrite steps of $t_0 \twoheadrightarrow t_\alpha$. The reduction $t_0 \twoheadrightarrow t_\alpha$ is *depth leftmost standard* iff for all $\beta < \lambda < \alpha$:

- if $p_\iota < p_\beta$ for some $\beta < \iota \leq \lambda$, then $p_\beta = p_\kappa \cdot p'_\beta$ with $p'_\beta \in \{q \in \mathcal{P}\mathrm{os}(l_\kappa) \mid \mathrm{root}(l_\kappa|_q) \in \Sigma\}$ and $\kappa = \min\{\gamma \in (\beta, \iota] \mid p_\beta > p_\gamma\}$, and otherwise
- either $|p_\beta| < |p_\lambda|$ or $|p_\beta| = |p_\lambda|$ and p_β occurs to the left of or is equal to p_λ.

Roughly, the above states that the position p_β should either (a) occur in the redex pattern of the first redex that occurs at a position above it or (b) occur above or to the left of any redex that is contracted later.

▶ **Example 4.13.** Consider the term $f(a, a)$ and the rewrite rule $a \to h(a)$ from Example 4.3. The following reduction is depth standard:

$$f(a,a) \to f(h(a),a) \to f(h(a),h(a)) \to \cdots \to f(h^n(a),h^n(a))$$
$$\to f(h^{n+1}(a),h^n(a)) \to f(h^{n+1}(a),h^{n+1}(a)) \to \cdots f(h^\omega,h^\omega)\,.$$

Note that the second clause of Definition 4.12 induces a total order on positions: We first order by depth and next we order from left to right. Hence, as the definition considers every initial $t_0 \twoheadrightarrow t_\lambda$, it follows for each $s \twoheadrightarrow t$ that there exists *at most* one depth leftmost standard reduction from s to t. Moreover, we have the following.

▶ **Lemma 4.14.** *Every depth leftmost standard reduction is parallel standard.*

Proof. Immediate by transfinite induction over the steps from the depth leftmost standard reduction, where the first clause of Definition 4.12 implies the second clause of Definition 4.2 and where the second clause of Definition 4.12 implies the first clause of Definition 4.2. ◀

Finally, we have:

▶ **Theorem 4.15.** *Let $s \twoheadrightarrow t$ in a left-linear iTRS. There exists a depth leftmost standard reduction of length at most ω from s to t.*

Proof (Sketch). By Theorem 4.10, there exists a parallel standard reduction from s to t. This parallel standard reduction is not necessarily depth leftmost standard, as certain redexes may violate the second clause of Definition 4.12. However, all these redexes will be parallel to each other and, hence, can be permuted without reservation (as long as we ensure that the finitely many redexes needed to create each redex are also permuted). The resulting reduction will be strongly convergent, as the second clause of Definition 4.12 only requires us to permute redexes that occur at lesser or equal depth than the redexes preceding them. ◀

5 Conclusion and Proposed Research

As shown, the obvious generalisation of the compression property — replacing ω by an arbitrary countable ordinal — does in principle not apply to iTRSs with non-left-linear rules. This result might be considered rather unexpected, as the standard counterexample to compression does satisfy such a property, albeit only for finite starting terms [5], and as does $\lambda\beta\eta$-calculus, the standard counterexample in the higher-order case [12].

Given the above observation and noting that infinitary rewriting is susceptible to a similar computational treatment as the real numbers in computable analysis [10], it seems we can no longer maintain a computational view of the compression property. Hence, we provide a reinterpretation, viz. that compression is a degenerate case of standardisation.

Given that compression and standardisation are closely related and that both have strong ties with notions related to the equivalence of reductions [8, 14], a further study of reduction equivalence in infinitary rewriting seems warranted for. Moreover, a pertinent question is whether our standardisation results can be used to simplify existing proofs in infinitary rewriting, e.g., the proof of confluence modulo hypercollapsing subterms [8, 7]. Finally, it would be interesting to investigate whether both the counterexamples to α-compression and the standardisation theorem extend to infinitary higher-order systems [11], where any counterexample to α-compression then employs a non-fully-extended rule instead of a non-left-linear rule.

Acknowledgements. The author wishes thank Jan Willem Klop, Yoshihito Toyama, and Jaco van de Pol for their continuing support. Moreover, acknowledgements go to the reviewers for their constructive feedback and to Jakob Grue Simonsen for encouraging the author to complete the paper. Finally, the author is indebted to Alastair Donaldson and the Department of Computing at Imperial College London for their financial support.

References

1 Søren Bjerg Andersen and Jakob Grue Simonsen. Term rewriting systems as topological dynamical system. In *Proc. 23rd Int'l Conf. on Rewriting Techniques and Applications (RTA 2012)*, volume 15 of *Leibniz International Proc. in Informatics*, pages 53–68, 2012.

2 André Arnold and Maurice Nivat. The metric space of infinite trees. Algebraic and topological properties. *Fundamenta Informaticae*, 3(4):445–476, 1980.

3 Michael Barr. Terminal coalgebras in well-founded set theory. *Theoretical Computer Science*, 114(2):299–315, 1993.

4 Inge Bethke, Jan Willem Klop, and Roel de Vrijer. Descendants and origins in term rewriting. *Information and Computation*, 159(1-2):59–124, 2000.

5 Nachum Dershowitz, Stéphane Kaplan, and David A. Plaisted. Rewrite, rewrite, rewrite, rewrite, rewrite, ... *Theoretical Computer Science*, 83(1):71–96, 1991.

6 Stefan Kahrs. Infinitary rewriting: Foundations revisited. In *Proc. 21st Int'l Conf. on Rewriting Techniques and Applications (RTA 2010)*, volume 6 of *Leibniz International Proceedings in Informatics*, pages 161–176, 2010.

7 Richard Kennaway and Fer-Jan de Vries. Infinitary rewriting. In Terese [13], Chapter 12, pages 668–711.

8 Richard Kennaway, Jan Willem Klop, Ronan Sleep, and Fer-Jan de Vries. Transfinite reductions in orthogonal term rewriting systems. *Information and Computation*, 119(1):18–38, 1995.

9 Jeroen Ketema. *Böhm-Like Trees for Rewriting.* PhD thesis, Vrije Universiteit, Amsterdam, 2006.

10 Jeroen Ketema and Jakob Grue Simonsen. Computing with infinite terms and infinite reductions. Unpublished manuscript.

11 Jeroen Ketema and Jakob Grue Simonsen. Infinitary Combinatory Reduction Systems. *Information and Computation*, 209(6):893–926, 2011.

12 Paula Severi and Fer-Jan de Vries. An extensional Böhm model. In *Proc. 13th Int'l Conf. on Rewriting Techniques and Applications (RTA 2002)*, volume 2378 of *Lecture Notes in Computer Science*, pages 159–173, 2002.

13 Terese, editor. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 2003.

14 Vincent van Oostrom and Roel de Vrijer. Equivalence of reductions. In Terese [13], Chapter 8, pages 301–474.

15 Klaus Weihrauch. *Computable Analysis: An Introduction.* Springer-Verlag, 2000.

Finite Models vs Tree Automata in Safety Verification

Alexei Lisitsa[1]

1 **Department of Computer Science**
University of Liverpool, UK
a.lisitsa@liverpool.ac.uk

Abstract

In this paper we deal with verification of safety properties of term-rewriting systems. The verification problem is translated to a purely logical problem of finding a finite countermodel for a first-order formula, which is further resolved by a generic finite model finding procedure. A finite countermodel produced during successful verification provides with a concise description of the system invariant sufficient to demonstrate a specific safety property.

We show the relative completeness of this approach with respect to the tree automata completion technique. On a set of examples taken from the literature we demonstrate the efficiency of finite model finding approach as well as its explanatory power.

1998 ACM Subject Classification I.2.3 Deduction and Theorem Proving, F.4.2 Grammars and Other Rewriting Systems, D.2.4 Software/Program Verification

Keywords and phrases term-rewriting systems, safety verification, first-order logic, finite model finding

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.225

Category Regular Research Paper

1 Introduction

The development of general automated methods for the verification of infinite-state and parameterized systems poses a major challenge. In general, such problems are undecidable, so one can not hope for the ultimate solution and the development should focus on the restricted classes of systems and properties.

In this paper we deal with a very general method for verification of *safety* properties of infinite-state systems which is based on a simple idea. If an evolution of a computational system is faithfully modelled by a derivation in a classical first-order logic then safety verification (non-reachability of unsafe states) can be reduced to the disproving of a first-order formula. The latter task can be (partially, at least) tackled by generic automated procedures searching for *finite* countermodels.

Such an approach to verification was originated in the research on formal verification of security protocols [28, 27, 13] and later has been extended to the wide classes of infinite-state and parameterised verification tasks. Completeness of the approach for particular classes of systems (lossy channel systems) and relative completeness with respect to general method of regular model checking has been established in [20] and [21] respectively.

Here we continue investigation of the boundaries of applicability of finite countermodels based method and are looking into verification of safety properties of term-rewriting systems (TRS). Term-rewriting systems provide with a general formalism for specification and verification of infinite-state systems. Several general automated methods for verification of safety properties of term-rewriting systems has been proposed and implemented [12, 9, 10] with the methods based on tree automata completion [12, 9] playing the major role.

We show that verification via finite countermodels (FCM) approach provides with a viable alternative to the methods based on the tree automata completion. We show the relative completeness of FCM with respect to the tree automata completion methods (TAC).

We illustrate it on a simple example taken from [11]. Consider the TRS $\mathcal{R} = \{f(x) \to f(s(s(x)))\}$ and assume that we want to prove that $f(a) \not\to^* f(s(a))$. In [11] a simple finite-state abstraction of the set of reachable terms expressed by the equation $E = \{s(s(x)) = x\}$ is explicitly added to the TRS and simple analysis of rewriting modulo E is proposed. In FCM approach, the same problem is translated into disproving of the first-order formula $\varphi_{\mathcal{R}} := (\forall x R(f(x), f(s(s(x)))) \to R(f(a), f(s(a))$. The intended meaning of the binary predicate R here is to encode the reachability relation for the TRS. The finite countermodel of $\varphi_{\mathcal{R}}$, having the size 2 (cardinality of the domain) and essentially representing the above abstraction, i.e. satisfying $s(s(x) = x$, can be found by an automated model finder, e.g. Mace4 in a fraction of a second.

On a series examples taken from the literature we demonstrate practical efficiency, the high degree of automation and the explanatory power of FCM approach using off-the shelf and state of the art implementation of a finite model finding procedure Mace4 (W. McCune).

The preliminary version of this paper has appeared as the report [22].

2 Preliminaries

In this paper we use standard terminology for first-order predicate logic and term-rewriting systems, and the for detailed accounts of these areas the reader is referred to [8] and to [2], respectively. We remind here only the concepts which we are going to use in the paper.

2.1 First-order Logic

The *first-order vocabulary* is defined as a finite set $\Sigma = \mathcal{F} \cup \mathcal{P}$ where $\mathcal{F}$ and $\mathcal{P}$ are the sets of functional and predicate symbols, respectively. Each symbol in Σ has an associated arity, and we have $\mathcal{F} = \cup_{i \geq 0} \mathcal{F}i$ and $\mathcal{P} = \cup_{i \geq 1} \mathcal{P}_i$, where $\mathcal{F}_i$ and $\mathcal{P}_i$ consist of symbols of arity i. The elements of $\mathcal{F}_0$ are also called *constants*.

First-order model over vocabulary Σ, or just a *model* is a pair $\mathcal{M} = \langle D, [\![\Sigma]\!]_D \rangle$ where D is a set called *domain* of $\mathcal{M}$ and $[\![\Sigma_D]\!]$ denotes the *interpretations* of all symbols from Σ in D. For a domain D and a function symbol f of arity $n \geq 1$ an interpretation of f in D is a function $[\![f]\!]_D : D^n \to D$. For a constant c its interpretation $[\![c]\!]_D$ is an element of D. For a domain D and a predicate symbol P of arity n an interpretation of P in D is a relation of arity n on D, that is $[\![P]\!]_D \subseteq D^n$. The model $\mathcal{M} = \langle D, [\![\Sigma]\!]_D \rangle$ is called *finite* if D is a finite set.

We assume that the reader is familiar with the standard definitions of first-order *formula*, first-order *sentence*, satisfaction $\mathcal{M} \models \varphi$ of a formula φ in a model $\mathcal{M}$, deducibility (derivability) $\Phi \vdash \varphi$ of a formula φ from a set of formulae Φ, first-order *theory*, *equational* theory.

2.2 Term-rewriting systems and tree automata

To define a term-rewriting system we fix a finite set of functional symbols $\mathcal{F}$, each associated with an arity and a set of variables $\mathcal{X}$. $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and $\mathcal{T}(\mathcal{F})$ denote the set of terms and ground terms, respectively, defined in the standard way using $\mathcal{F}$ and $\mathcal{X}$. The set of variables of a term t is denoted by $Var(t)$. A substitution is a function $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$, which can be extended homomorphically in a unique way (and keeping the name) to $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}, \mathcal{X})$. Application of a substitution σ to a term t we denote by $t\sigma$.

A term-rewriting system $\mathcal{R}$ is a set of rewrite rules $l \to r$ where $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \notin \mathcal{X}$ and $Var(r) \subseteq Var(l)$. The notion of a *subterm* is defined in a standard way. *One-step rewriting relation* $\Rightarrow_{\mathcal{R}} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}, \mathcal{X})$ is defined as follows: $t_1 \Rightarrow_{\mathcal{R}} t_2$ holds iff t_2 is obtained from t_1 by replacement of a subterm $l\sigma$ of t_1 with a subterm $r\sigma$ for some rewriting rule $(l \to r)$ in $\mathcal{R}$ and some substitution σ. The reflexive and transitive closure $\Rightarrow_{\mathcal{R}}$ is denoted by $\Rightarrow^*_{\mathcal{R}}$.

Given a set E of $\mathcal{F}$-equations, $\mathcal{T}_E$ denotes a set of equivalent classes of ground $\mathcal{F}$-terms modulo the equations E. E-equivalence class of a term t is denoted by $[t]_E$. Given a term rewriting system $\mathcal{R}$ and an equational theory given by a set of equations E we denote by $\Rightarrow_{\mathcal{R},E}$ one step rewriting relation modulo E, that is $t \Rightarrow_{\mathcal{R},E} t'$ iff $\exists \tau \in [t]_E \; \exists \tau' \in [t']_E \; \tau \Rightarrow_{\mathcal{R}} \tau'$.

Let Q be a finite set of symbols called *states* which we formally treat as functional symbols of arity 0 (constants). We assume $Q \cap \mathcal{F} = \emptyset$. Elements of $\mathcal{T}(\mathcal{F} \cup Q)$ are called *configurations*.

▶ **Definition 2.1.** (Transitions) A *transition* is a rewrite rule $c \to q$, where c is a configuration, i.e. $c \in \mathcal{T}(\mathcal{F} \cup Q)$, and $q \in Q$. A *normalized* transition is a transition $c \to q$ where $c = f(q_1, \ldots, q_n)$, f is a functional symbol of arity n from $\mathcal{F}$, $q, q_1, \ldots q_n \in Q$. An ϵ-*transition* $c \to q$ is such that $c \in Q$.

▶ **Definition 2.2.** (Tree automata) A (bottom-up, non-deterministic, finite) tree automaton is a quadruple $\mathcal{A} = \langle \mathcal{F}, Q, Q_f, \Delta \rangle$, where $Q_f \subseteq Q$ is a set of final (accepting) states and Δ is a set of normalized transitions and of ϵ-transitions. A tree automaton is deterministic if there are no two transitions in Δ with the same left-hand side.

Transitions Δ of $\mathcal{A}$ induce the *rewriting relation* on $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ which is denoted by $\Rightarrow_\Delta$ or $\Rightarrow_{\mathcal{A}}$.

▶ **Definition 2.3.** (Recognized language) The tree language recognized by $\mathcal{A}$ in a state q is $L(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \Rightarrow^*_{\mathcal{A}} q\}$. The language recognized by $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \cup_{q \in Q_f} \mathcal{L}(\mathcal{A}, q)$.

▶ **Example 2.4.** (Tree automaton and recognized language) Let $\mathcal{F} = \{f, a, b\}$ and $\mathcal{A} = \langle \mathcal{F}, Q, Q_f, \Delta \rangle$, where $Q = \{q_1, q_2\}$, $Q_f = \{q_1\}$, and $\Delta = \{f(q_1) \to q_1, a \to q_1, b \to q_2, q_2 \to q_1\}$. Then $\mathcal{L}(\mathcal{A}, q_1) = \mathcal{T}(f, a, b)$, that is the set of all terms build on $\{f, a, b\}$, and $\mathcal{L}(\mathcal{A}, q_2) = \{b\}$.

Deterministic bottom-up tree automata have the same expressive power as non-deterministic bottom-up tree automata, that is they recognize the same classes of term languages. In what follows we assume that automata are *deterministic*, unless otherwise specified.

3 Safety via finite countermodels

3.1 Basic verification problem

The main verification problem we consider in this paper is as follows.

▶ **Problem 1.**
Given: *Tree automata $\mathcal{A}_I$ and $\mathcal{A}_U$, a term-rewriting system $\mathcal{R}$*
Question: *Does $\forall t_1 \in \mathcal{L}(\mathcal{A}_I) \; \forall t_2 \in \mathcal{L}(\mathcal{A}_U) \; t_1 \not\Rightarrow^*_{\mathcal{R}} t_2$ hold?*

In applications, we assume that the states of a computational system to be verified are represented by terms, the system evolution (computation) is represented by $\mathcal{R}$; tree automata $\mathcal{A}_I$ and $\mathcal{A}_U$ provide with finitary specifications of the (infinite, in general) sets of allowed *initial* states and the sets of *unsafe* states, presented by $\mathcal{L}(\mathcal{A}_I)$ and $\mathcal{L}(\mathcal{A}_U)$, respectively. Under such assumptions, safety of the system is equivalent to the positive answer on the question of the Problem 1.

3.2 Translation of the basic verification problem

In this subsection we show how to reduce the basic verification problem to the problem of disproving of a formula from classical first-order predicate logic.
First, we define the translation Φ_R of a term-rewriting system $\mathcal{R}$ over $\mathcal{T}(\mathcal{F},\mathcal{X})$ into a set of first-order formulae in the vocabulary $\mathcal{F}\cup\{R\}$, where R is a new binary predicate symbol. Let $\Phi_\mathcal{R} = \Phi^r_\mathcal{R}\cup\Phi_\mathcal{F}$, where $\Phi^r_\mathcal{R} = \{R(l,r) \mid (l\to r)\in\mathcal{R}\}$ and $\Phi_\mathcal{F}$ is the set of the following formulae, which are all assumed to be universally closed and where $x_1,\dots x_i,\dots x_n,x_i^{'}$ are distinct variables:

1. $R(x,x)$ **reflexivity axiom**
2. $R(x,y)\wedge R(y,z)\to R(x,z)$ **transitivity axiom**
3. $R(x_i,x_i^{'})\to R(f(x_1,\dots,x_i,\dots x_n),f(x_1,\dots,x_i^{'},\dots x_n))$ for every n-ary functional symbol f from $\mathcal{F}$ and every position i: $1\le i\le n$ **congruence axioms**

Under such a translation first-order derivability faithfully models rewriting in $\mathcal{R}$ as the following proposition shows.

▶ **Proposition 3.1.** *For ground terms t_1, $t_2\in\mathcal{T}(\mathcal{F})$ if $t_1\Rightarrow^*_\mathcal{R} t_2$ then $\Phi_\mathcal{R}\vdash R(t_1,t_2)$.*

Proof. Due to the transitivity of R specified in $\Phi_\mathcal{R}$ it is sufficient to show that if $t_1\Rightarrow_\mathcal{R} t_2$ then $\Phi_\mathcal{R}\vdash R(t_1,t_2)$. Assume $t_1\Rightarrow t_2$ then t_2 is obtained from t_1 by the replacement of some subterm $l\sigma$ of t_1 with a subterm $r\sigma$ for some $(l\to r)\in\mathcal{R}$ and some substitution σ. Consider two sequences of subterms $\tau_0 = l\sigma,\tau_1,\dots,\tau_k = t_1$ and $\rho_0 = r\sigma,\rho_1,\dots,\rho_k = t_2$ with the property that τ_i is an immediate subterm of τ_{i+1} within t_1 and ρ_i is an immediate subterm of ρ_{i+1} within t_2, $i = 0,\dots,k$. Then we show by easy induction on i that $\Phi_\mathcal{R}\vdash R(\tau_i,\rho_i)$ for $i = 0,\dots,k$. Indeed, for $i = 0$ we have $R(\tau_0,\rho_0)\equiv R(l\sigma,r\sigma)$ is a ground instance of $R(l,r)\in\Phi^r_\mathcal{R}$ and therefore $\Phi_\mathcal{R}\vdash R(\tau_0,\rho_0)$. For the step of induction, assume $\Phi_\mathcal{R}\vdash R(\tau_i,\rho_i)$. Notice that by construction of sequences of τ's and ρ's τ_{i+1} and ρ_{i+1} should have the same outermost functional symbol f and coincide everywhere apart of subterms τ_i and ρ_i. Let $\tau_{i+1} = f(\dots,\tau_i,\dots)$ and $\rho_{i+1} = f(\dots,\rho_i,\dots)$. Then we have $R(\tau_i,\rho_i)\to R(\tau_{i+1},\rho_{i+1})$ is a ground instance of one of the formulae in $\Phi^r_\mathcal{R}$. So we have $\Phi_\mathcal{R}\vdash R(\tau_i,\rho_i)\to R(\tau_{i+1},\rho_{i+1})$ and by inductive assumption $\Phi_\mathcal{R}\vdash R(\tau_i,\rho_i)$. It follows $\Phi_\mathcal{R}\vdash R(\tau_{i+1},\rho_{i+1})$. The induction step is completed. We have $\Phi_\mathcal{R}\vdash R(\tau_k,\rho_k)$, which is $\Phi_\mathcal{R}\vdash R(t_1,t_2)$ ◀

Now we define a first-order translation of a tree automaton.

Let $\mathcal{A} = \langle\mathcal{F},Q_I,Q_f,\Delta\rangle$ be a tree automaton. let $\Sigma_\mathcal{A}$ be the following first-order vocabulary:

- constants for all elements of Q;
- all functional symbols from $\mathcal{F}$;
- a binary predicate symbol R;

Let $\Phi_{\mathcal{A}}$ to be the set of first-order formulae in vocabulary $\Sigma_{\mathcal{A}}$ defined as $\Phi_{\mathcal{A}} = \Phi_{\Delta} \cup \Phi_{\mathcal{F}}$, where $\Phi_{\Delta} = \{R(c,q) \mid (c \to q) \in \Delta\}$ and $\Phi_{\mathcal{F}}$ is as defined above.

As the following proposition shows first-order logic derivations from $\Phi_{\mathcal{A}}$ faithfully simulate the work of the automaton $\mathcal{A}$

▶ **Proposition 3.2.** *(Adequacy of automata translation)*
If $t \in \mathcal{L}_{\mathcal{A}}$ then $\Phi_{\mathcal{A}} \vdash \vee_{q \in Q_f} R(t,q)$

Proof. The statement of the proposition follows immediately from Definitions 2.2 and 2.3 and Proposition 3.1.

◀

Now we are ready to define the translation of the basic verification problem. Assume we are given an instance $P = \langle \mathcal{A}_I, \mathcal{R}, \mathcal{A}_U \rangle$ of Problem 1, with a term-rewriting system $\mathcal{R}$ over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and tree automata $\mathcal{A}_I = \langle \mathcal{F}, Q_I, Q_{f_I}, \Delta_I \rangle$, $\mathcal{A}_U = \langle \mathcal{F}, Q_U, Q_{f_U}, \Delta_U \rangle$. Assume also (without loss of generality) that sets $\mathcal{F}$, Q_I and Q_U are disjoint.

We define translation of P as $\Phi_P = \Phi_{\mathcal{A}_I} \cup \Phi_{\mathcal{A}_U} \cup \Phi_{\mathcal{R}}$. By the above definitions we then also have $\Phi_P = \Phi_{\mathcal{F}} \cup \Phi_{\Delta_I} \cup \Phi_{\Delta_U} \cup \Phi^r_{\mathcal{R}}$. We further define the translation of (negation of) correctness condition from P as a formula $\psi_P = \exists x \exists y \vee_{q_i \in Q_I, q_u \in Q_U} R(x, q_i) \wedge R(x,y) \wedge R(y, q_u)$.

The following proposition and corollary serves as a formal underpinning of the proposed verification method.

▶ **Proposition 3.3.** *(Correctness of the translation)*
Let P be an instance of the basic verification problem as detailed above. Then if P has a negative answer then $\Phi_P \vdash \psi_P$

Proof. The statement of the proposition immediately follows from Definitions 2.2 and 2.3 and Propositions 3.1 and 3.2. ◀

By contraposition we have the following

▶ **Corollary 3.4.** *If $\Phi_P \not\vdash \psi_P$ the instance P has a positive answer and the safety property holds.*

3.3 FCM method

By FCM (finite countermodels) verification method we understand the following. Given an instance $P = \langle \mathcal{A}_I, \mathcal{R}, \mathcal{A}_U \rangle$ of the basic verification problem, translate it into a set of first-order formulae Φ_P and a formula ψ_P as described above. Then apply a generic finite model finding procedure to find a countermodel for $\Phi_P \to \psi_P$. If a countermodel is found the safety property is established and the instance P has got a positive answer.

3.4 Relative completeness

In this section we show the *relative completeness* of FCM with respect to verification methods based on tree automata completion techniques (TAC). More precisely, we show that if safety of TRS can be demonstrated by TAC, it can be demonstrated by FCM too.

Given an instance P of basic verification problem (Problem 1) verification by TAC approach would proceed as follows. Starting from $\mathcal{A}_{\mathcal{I}}$ and $\mathcal{R}$ completion procedure yields an automaton $\mathcal{A}^*$ wich describes, in general, an *overapproximation* of the set of terms reachable in $\mathcal{R}$ from $\mathcal{L}(\mathcal{A}_{\mathcal{I}})$), that is $\mathcal{L}(\mathcal{A}^*) \supseteq \{t \mid \exists t_0 \in \mathcal{L}(\mathcal{A}_{\mathcal{I}}) \ \ t_0 \to^*_{\mathcal{R}} t\}$. Further, the check of whether $\mathcal{L}(\mathcal{A}^*) \cap \mathcal{L}(\mathcal{A}_{\mathcal{U}}) = \emptyset$ is performed and, if it holds, the safety is established.

Exact description of the set of all reachable terms in a term-rewriting system by a tree automaton is not always possible. The main direction in the development of TAC methods is a development of more efficient and more precise approximations methods.

▶ **Theorem 3.5.** *Let $P = \langle \mathcal{A}_I, \mathcal{A}_U, \mathcal{R}\rangle$ be a basic verification problem and there exists a tree automaton $\mathcal{A}^* = \langle \mathcal{F}, Q^*, Q^*_f, \Delta^*\rangle$ such that $\mathcal{L}(\mathcal{A}^*) \supseteq \{t \mid \exists t_0 \in \mathcal{L}(\mathcal{A_I})\ \ t_0 \to^*_{\mathcal{R}} t\}$ and $\mathcal{L}(\mathcal{A}^*) \cap \mathcal{L}(\mathcal{A_U}) = \emptyset$. Then there exists a finite model $\mathcal{M}$ such that $\mathcal{M} \models \Phi_P \wedge \neg\psi_P$ (i.e $\mathcal{M}$ is a countermodel for $\Phi_P \to \psi_P$).*

Proof. Assume the conditions of the theorem hold. Define the domain D of the required model: $D = Q_I^{\perp} \times Q_*^{\perp} \times Q_U^{\perp}$, where $Q_I^{\perp} = Q_I \cup \{\perp\}$.

Define interpretations of constants $[\![c]\!] = \langle a_I, a_*, a_U\rangle$, where $a_x = q$ if $(c, q) \in \Delta_x$, or $a_x = \perp$ otherwise, $x \in \{I, *, U\}$. For a functional symbol f of arity $n \geq 1$ define its interpretation $[\![f]\!] : D^n \to D$ as $[\![f]\!](\langle a^1_I, a^1_*, a^1_U\rangle, \ldots, \langle a^n_I, a^n_*, a^n_U\rangle) = \langle a_I, a_*, a_U\rangle$, where for all $x \in \{I, *, U\}$, either $(f(a^1_x, a^2_x, \ldots a^n_x) \to a_x) \in \Delta_x$, or $a_x = \perp$, otherwise.

Once we defined the interpretations of all functional symbols (including constants) any ground term t gets its interpretation $[\![t]\!] \in D$ in a standard way. Then it is an easy consequence of definitions that $[\![t]\!]$ is a triple of states of automata $\mathcal{A}_I$, $\mathcal{A}_*$, $\mathcal{A}_U$, respectively, into which they get working on the input t. More formally, if $[\![t]\!] = \langle a_I, a_*, a_U\rangle$, then for all $x \in \{I, *, U\}$ either $t \Rightarrow^*_x a_x \in Q_x$, or there is no such $q \in Q_x$ that $t \Rightarrow^*_x q$, and then $t \Rightarrow^*_x a_x = \perp$.

Define the interpretation $[\![R]\!] \subseteq D \times D$ of R as follows.

$$[\![R]\!] = \{\langle [\![t_1]\!], [\![t_2]\!]\rangle \mid t_1, t_2 \text{ are ground in } D, t_1 \Rightarrow^* t_2\}$$

where $\Rightarrow$ denotes $\Rightarrow_{\mathcal{R}} \cup \Rightarrow_{\Delta_I} \cup \Rightarrow_{\Delta_U}$.

Now we are going to show that in a such defined model $\mathcal{M}$ we have $\Phi_P \wedge \neg\psi_P$ satisfied. Recall $\Phi_P = \Phi_{\mathcal{F}} \cup \Phi_{\Delta_I} \cup \Phi_{\Delta_U} \cup \Phi^r_{\mathcal{R}}$.

We have

- $\mathcal{M} \models \Phi_{\mathcal{F}}$ (by definition of rewriting and definition of $[R]$)
- $\mathcal{M} \models \Phi_{\Delta_I} \cup \Phi_{\Delta_U} \cup \Phi^r_{\mathcal{R}}$ (by definition of $[R]$)

To show $\mathcal{M} \models \neg\psi_P$ assume the opposite i.e $\mathcal{M} \models \psi_P$ that is $\mathcal{M} \models \exists x \exists y \vee_{q_i \in Q_I, q_u \in Q_U} R(x, q_i) \wedge R(x, y) \wedge R(y, q_u)$. That means there are $a, b \in D$ such that $(a, [\![q_i]\!]) \in [\![R]\!]$, $(a, b) \in [\![R]\!]$, $(b, [\![q_u]\!]) \in [\![R]\!]$. Consider the ground terms τ_1 and τ_2 such that $[\![\tau_1]\!] = a$ and $[\![\tau_2]\!] = b$. We have $\tau_1 \in \mathcal{L}(\mathcal{A}_I)$, $\tau_1 \Rightarrow^* \tau_2$, $\tau_2 \in \mathcal{L}(\mathcal{A}_U)$. It follows that $\tau_2 \in \mathcal{L}(\mathcal{A}^*) \cap \mathcal{L}(\mathcal{A_U})$ which contradicts to the assumption of the theorem on emptiness of $\mathcal{L}(\mathcal{A}^*) \cap \mathcal{L}(\mathcal{A_U})$. ◀

▶ **Note 1.** *The above model construction serves only the purpose of proof and it is not efficient in practical use of the method. Instead we assume that the task of model construction is delegated to a generic finite model building procedure.*

3.5 Finite models as invariants

Assume that for a basic verification problem $P = \langle \mathcal{A}_I, \mathcal{A}_U, \mathcal{R}\rangle$, a model $\mathcal{M}$ such that $\mathcal{M} \models \Phi_P \wedge \neg\psi_P$ is found. Then not only the safety is established, the model itself provides with a finite description of an invariant of a TRS sufficient to prove the safety. Define the following subsets of a domain D of $\mathcal{M}$.

- $I = \{[\![t]\!] \mid t \in \mathcal{L}(\mathcal{A_I})\}$
- $U = \{[\![t]\!] \mid t \in \mathcal{L}(\mathcal{A_U})\}$
- $Reach = \{[\![t]\!] \mid \exists t_0 \in \mathcal{L}(\mathcal{A_I})\ \ t_0 \to^*_{\mathcal{R}} t\}$
- $I \bullet [\![R]\!] = \{y \mid \exists x \in I \wedge (x, y) \in [\![R]\!]\}$ (where $[\![R]\!]$ is an interpretation of R in $\mathcal{M}$)

Then it is easy consequence of definitions of Φ_P and ψ_P that $Reach \subseteq I \bullet [\![R]\!]$ and $I \bullet [\![R]\!] \cap U = \emptyset$. Thus $I \bullet [\![R]\!]$ is a finite invariant set which subsumes the interpretations of all reachable terms and at the same time is disjoint with the set of interpretations of all unsafe terms. For a finite model $\mathcal{M}$ both conditions can be verified by a straightforward induction, providing thereby the translation of FCM verification into an inductive proof of a safety property.

3.6 Variations on a theme

Theorem 3.5 provides with a lower bound for the verifying power of FCM method applied to a basic verification problem. In this section we consider practically important variations of the basic verification problem which allow simplified translations and more efficient verification.

3.6.1 Finitely based sets of terms

In many cases of safety verification tasks for TRS the sets of initial and/or unsafe terms are given not by tree automata, but rather described as the sets of ground instances of terms from a given finite set of terms. More precisely, let B be a finite set of terms in a vocabulary $\mathcal{F}$ and $g(B) = \{\tau \mid \exists t \in B \wedge \tau = t\theta; \theta \text{ is ground }\}$. It is easy to see that for the finite B $g(B)$ is a regular set.

Consider the following modification of the basic verification problem.

▶ **Problem 2.**

Given: *Finite sets of terms B_I and B_U, a term-rewriting system $\mathcal{R}$*

Question: *Does $\forall t_1 \in g(B_I)\ \forall t_2 \in g(B_U)\ t_1 \not\to^*_{\mathcal{R}} t_2$ hold?*

Let $P = \langle B_I, \mathcal{R}, B_U \rangle$ be an instance of the Problem 2.

The translation $\Phi_{\mathcal{R}}$ of the term rewriting system $\mathcal{R}$ is defined in 3.2.

The translation of (negation of) correctness condition from P is defined as $\psi_P = \exists \bar{x} \bigvee_{t_1 \in g(B_I), t_2 \in g(B_U)} R(t_1, t_2)$.

Now we have the following analogue of Proposition 3.3

▶ **Proposition 3.6.** *(Correctness of the translation)*

Let P be an instance of the basic verification problem as detailed above. Then if P has a negative answer then $\Phi_{\mathcal{R}} \vdash \psi_P$

3.6.2 Outermost rewriting

Another simplification of the translation may come from the restrictions on the rewriting strategies in TRSs. If rewriting can only be applied at the outer level, i.e. redex can be only the whole term, not its proper subterm, then the first-order translation of an TRS can be simplified by using unary reachability predicate $R(-)$ instead of binary $R(-,-)$. The intended meaning of $R(t)$ is "term t is reachable from some of the initial terms (using outermost rewriting)". The translation of a term rewriting system $\mathcal{R}$ becomes especially simple in this case: $\Phi_{\mathcal{R}} = \{R(t) \to R(t') \mid (t \to t') \in \mathcal{R}\}$.

We omit the obvious further details of translation (e.g. on specification of the sets of initial and unsafe terms) as well as the statement on its correctness and rather refer the reader to the Example 6.2.

4 Rewriting modulo theories

Proposed FCM method is very flexible and can be naturally extended to the rewriting modulo equational theories. We consider here only the case of basic verification problem. The extensions of other variants (such as Problem 2, or as in subsection 3.6.2) can be dealt with in a similar way.

▶ **Problem 3.**

Given: *Tree automata $\mathcal{A}_I$ and $\mathcal{A}_U$, a term-rewriting system $\mathcal{R}$, and a set of equations E*

Question: *Does $\forall t_1 \in \mathcal{L}(\mathcal{A}_I)\ \forall t_2 \in \mathcal{L}(\mathcal{A}_U)\ [t_1] \not\Rightarrow^*_{\mathcal{R},E} [t_2]$ hold?*

Using notation introduced in subsection 3.2 we now have

▶ **Proposition 4.1.** *For ground terms $t_1, t_2 \in \mathcal{T}(\mathcal{F})$ if $[t_1] \Rightarrow^*_{\mathcal{R},E} [t_2]$ then $\Phi_\mathcal{R} \cup E \vdash R(t_1, t_2)$.*

Proof. (Hint) The proof follows the proof of Proposition 3.1 using the following straightforward statement. If for some ground terms t_1, t_2 $\Phi_\mathcal{R} \cup E \vdash R(t_1, t_2)$ then for any $t'_1 \in [t_1]$ and $t'_2 \in [t_2]$ $\Phi_\mathcal{R} \cup E \vdash R(t'_1, t'_2)$. ◀

Based on that we have an equational analogue of Proposition 3.3 supporting applicability of FCM for proving the safety of TRS modulo equational theories:

▶ **Proposition 4.2.** *For the instance P of the above verification problem if P has a negative answer then $\Phi_{\mathcal{A}_I} \cup \Phi_{\mathcal{A}_U} \cup \Phi_\mathcal{R} \cup E \vdash \psi_P$*

Example 6.3 illustrates the use of FCM for equational rewriting. Since the outermost rewriting strategy is assumed in this Example the simplified translation with unary reachability predicate R is used.

5 Approximation by symmetric closure

In theory the application of FCM method is simple - apply finite model builder to the first-order encoding of the problem and wait. If there is a finite countermodel it will be eventually found if a complete model builder is used. In practice, however, it may take a long time to find a countermodel. To accelerate the search one may use the approximation of reachability relation $\Rightarrow_\mathcal{R}$ ($\Rightarrow_{\mathcal{R},E}$) by its symmetric closure $\Leftrightarrow_\mathcal{R}$ ($\Leftrightarrow_{\mathcal{R},E}$). It is obvious that if the safety holds for the original problem, it may be lost after the symmetric closure is applied. If however the safety is shown for the problem with the symmetric closure it holds for the original problem too. To take an account of the symmetric closure it is sufficient to add to the first-order translation of the basic verification problem just one formula $R(x, y) \rightarrow R(y, x)$. Notice that after such a modification an interpretation of R in any model is an equivalence relation which is a congruence. It follows that if a countermodel searched in FCM method exists then there exists a countermodel where R is interpreted by the *equality* relation. This observation allows to simplify the translation of such modified problem further. Assuming that a sound model finder procedure for the first-order logic with equality is used, in the first-order translation all occurrences of R can be replaced with equality and reflexivity, transitivity and congruence axioms can be dropped. Example 6.4 demonstrates the use of such a technique.

6 Experiments

In this section we present four examples of application of FCM method for safety verification and compare the results with the results of alternative methods reported in the literature.

In the experiments we used the finite model finder Mace4[24] within the package Prover9-Mace4, Version 0.5, December 2007. The system configuration used in the experiments: Microsoft Windows XP Professional, Version 2002, Intel(R) Core(TM)2 Duo CPU, T7100 @ 1.8Ghz 1.79Ghz, 1.00 GB of RAM. The time measurements are done by Mace4 itself, upon completion of the model search it communicates the CPU time used.

6.1 Parity of n^2

▶ **Example 6.1.**

The following verification task is taken from [12, 10].

Let $P_{n^2} = \langle \mathcal{A}_I, \mathcal{R}, \mathcal{A}_U \rangle$ be an instance of basic verification task. Term rewriting system $\mathcal{R}$ consists of the following rewriting rules

- $plus(0, x) \to x$
- $plus(s(x), y) \to s(plus(x, y))$
- $times(0, x) \to 0$
- $times(s(x), y) \to plus(y, times(x, y))$
- $square(x) \to times(x, x)$
- $even(0) \to true$
- $even(s(0)) \to false$
- $even(s(x)) \to odd(x))$
- $odd(0) \to false$
- $odd(s(0)) \to true$
- $odd(s(x)) \to even(x)$
- $even(square(x)) \to odd(square(s(x)))$
- $odd(square(x)) \to even(square(s(x)))$

The tree automaton $\mathcal{A}_I$ recognizes the set of initial terms. It has the set of states $Q_I = \{s0, s1, s2\}$, the set

of the final states $Q_{I_f} = \{s0\}$ and the set of rewriting rules $\Delta_I = \{even(s1) \to s0, square(s2) \to s1, 0 \to s2\}$ It is easy to see that $\mathcal{L}(\mathcal{A}_I) = \{even(square(0))\}$

The tree automaton $\mathcal{A}_U$ recognizes the set of unsafe terms. It has the set of states $Q_U = Q_{U_f} = \{q0\}$ and the set of rewriting rules $\Delta_U = \{false \to q0\}$.

So the question of the verification problem P_{n^2} is whether $false$ is reachable from $even(square(0))$.

Denote by Φ_P the first-order translation of P_{n^n} as defined in Section 3.2. The formula $\psi_P : \exists x \exists y (R(x, s0) \wedge R(x, y) \wedge R(y, q0)$ expresses the negation of correctness condition.

The finite model finder Mace4 has found a finite countermodel for $\Phi_P \to \psi_P$ (i.e a finite model for $\Phi_P \wedge \neg\psi_P$) in 0.03s. The domain D of the model is a two element set $\{0, 1\}$. Interpretations of constants: $[\![f]\!] = [\![q0]\!] = [\![s1]\!] = [\![s2]\!] = 0$; $[\![s0]\!] = [\![t]\!] = 1$. Interpretations of functions: $[\![even]\!](0) = 1$, $[\![even]\!](1) = 0$; $[\![odd]\!](0) = 0$, $[\![odd]\!](1) = 1$; $[\![s]\!](0) = 1$, $[\![s]\!](1) = 0$; $[\![square]\!](0) = 0$; $[\![square]\!](1) = 1$; $[\![plus]\!]$(x,y) $= (x + y) mod 2$; $[\![times]\!]$(x,y) $= x \times y$. Interpretation of reachability relation: $[\![R]\!] = \{(0, 0), (1, 1)\}$.

Notice that verification is done here automatically. This can be contrasted with the verification of the same system by a tree completion algorithm implemented in Timbuk system [9], where an user interaction was required to add an approximation equation $s(s(x)) = x$

manually. In [10] an automated verification of the same system was reported using Horn Clause approximation technique. The system was specified as a Horn Clause program and the verification followed by producing a model for the program which contained 53 elements. The above model produced by Mace4 within FCM approach provides with much more concise explanation of why the safety holds: interpretation of any ground term (0 or 1) is an invariant for reachability in TRS, $[\![even(square(0))]\!] = 1$ and $[\![f]\!] = 0$.

6.2 Readers-writers system verification

In this subsection we consider the example of a readers-writers system verification taken from [6, 11].

▶ **Example 6.2.**

In the TRS specifying the system the only *outermost* rewriting is possible, so for the translation we use *monadic* reachability predicate. Furthermore, both the set of initial terms and the set of unsafe terms are finitely based. The vocabulary consists the constant 0, unary functional symbol s (for successor) and binary functional symbol *state*.

The rules are as follows

- $state(0, 0) \to state(0, s(0))$
- $state(x, 0) \to state(s(x), 0)$
- $state(x, s(y)) \to state(x, y)$
- $state(s(x), y) \to state(x, y)$

The set of initial terms is $I = \{state(0, 0)\}$. The set of unsafe terms U is finitely based with the base $B = \{state(s(x), s(y)), state(x, s(s(y)))\}$. The first-order translation Φ consists the conjunction of the following formulae

- $R(state(0, 0))$
- $R(state(0, 0)) \to R(state(0, s(0)))$
- $R(state(x, 0)) \to R(state(s(x), 0))$
- $R(state(x, s(y))) \to R(state(x, y))$
- $R(state(s(x), y)) \to R(state(x, y))$

The formula $\psi \equiv \exists x \exists y R(s(x), s(y)) \vee R(x, s(s(y)))$ expresses the negation of the correctness condition.

The system can be then successfully verified by an FCM method. The search for the countermodel for $\Phi \to \psi$ took 0.01s and the model found is as follows.

The domain D of the model is a three element set $\{0, 1, 2\}$; $[\![s]\!](0) = 1$, $[\![s]\!](1) = 2$, $[\![s]\!](2) = 2$; $[\![R]\!] = \{(0, 0), (0, 1), (1, 0), (2, 0)\}$.

Notice that no additional information is needed for FCM method to automatically verify the reader-writer system. That may be contrasted with the verification using tree automata completion approach (Timbuk 3.0 system), reported in [11] where an equational abstraction rule $s(s(x)) = s(s(0))$ should be manually added to the TRS for the successful verification.

6.3 Glass replacement puzzle

▶ **Example 6.3.**

The following verification example is taken from [17], where it has been formalized using higher-order rewriting by simply-typed term rewriting systems (STRS) and solved by using higher-order Knuth-Bendix completion procedure. We use first-order term rewriting formalization of the same problem .

Assume a sequence of sake, whisky and beer glasses. The sequence can be modified by application of the following glass-replacement rules.

- A sake glass may be inserted to the left of a beer glass. A sake glass having a beer glass to its right may be removed.
- A sake glass and whisky glass may be added to the left and right, respectively, of an arbitrary contiguou s subsequence of glasses. The reverse operation may also be performed.

The verification problem is to show that starting from *sake-whisky-whisky-whisky* sequence and applying the replacement rules above one can not reach the sequence *sake-whisky-whisky-beer*.

We formalize the problem by the term-rewriting system $\mathcal{R}$ consisting the following rewriting rule

- $x * (sake * (beer * y)) \to x * (beer * y)$
- $x * (beer * y) \to x * (sake * (beer * y))$
- $v * (sake * (x * (whisky * y))) \to v * (x * y)$
- $v * (x * y) \to (v * (sake * (x * (whisky * y)))$

where $sake$, $beer$, $whisky$ are constants, and $*$ is a binary associative term constructor (we use it in infix notation). Associativity of $*$ means we consider rewriting modulo equational theory $E = \{(x * y) * z = x * (y * z)\}$.

First-order translation Φ consists the following formulae:

- $(x * y) * z = x * (y * z)$
- $P(x * (sake * (beer * y))) \to P(x * (beer * y))$
- $P(x * (beer * y)) \to P(x * (sake * (beer * y)))$
- $P((v * (sake * (x * (whisky * y)))) \to P(v * (x * y))$
- $P(v * (x * y)) \to P(v * (sake * (x * (whisky * y))))$

Now, to resolve the puzzle it is sufficient to show $\Phi \wedge P(sake * (whisky * (whisky * whisky))) \not\vdash P(sake * (whisky * (whisky * beer)))$. A countermodel of size 2 is found by finite model finder MACE4 in 0.01sec.

6.4 Reverse function

In this section we consider a verification problem from [12]. The problem here is to show that list reverse function satisfies the following property: if in a list all symbols 'a' are before all symbols 'b' then after reversing there are no 'a' before 'b'.

▶ **Example 6.4.**

Vocabulary $\mathcal{F}$ consists of one 0-ary functional (constant) symbol 0 and three binary symbols app, $cons$, rev.

The automaton recognizing is initial terms is defined as $\mathcal{A}_I = \langle \mathcal{F}, Q_I, Q_{f_I}, \Delta_I \rangle$, where $\mathcal{F}$ is as defined above; $Q_I = \{qrev, qlab, qlb, qa, qb\}$; $Q_{f_I} = \{qrev\}$; Δ_I contains

- $rev(qlab) \to qrev$
- $cons(qa, qlab) \to qlab$
- $0 \to qlb$

- $a \to qa$
- $0 \to qlab$
- $cons(qa, qlb) \to qlab$
- $cons(qb, qlb) \to qlb$
- $b \to qb$

The automaton recognizing *unsafe* terms is defined as $\mathcal{A}_U = \langle \mathcal{F}, Q_U, Q_{f_U}, \Delta_U \rangle$, where $\mathcal{F}$ is as above; $Q_U = \{qlab1, qlb1, q1, qa, qb\}$, $Q_{f_U} = \{qlab1\}$; Δ_U contains

- $cons(qa, qlab1) \to qlab1$
- $cons(qa, qlb1) \to qlab1$
- $cons(qa, q1) \to q1$
- $a \to qa$
- $0 \to q1$
- $cons(qb, qlab1) \to qlab1$
- $cons(qb, q1) \to qlb1$
- $cons(qb, q1) \to q1$
- $b \to qb$

The term-rewriting system $\mathcal{R}$ consists of the following rules

- $app(0, x) \to x$
- $app(cons(x, y), z) \to cons(x, app(y, z))$
- $rev(0) \to 0$
- $rev(cons(x, y)) \to app(rev(y), cons(x, 0))$

First-order translation Φ_P consists of the following formulae.

- $R(rev(qlab), qrev)$
- $R(cons(qa, qlab), qlab)$
- $R(0, qlb)$
- $R(a, qa)$
- $R(0, qlab)$
- $R(cons(qa, qlb), qlab)$
- $R(cons(qb, qlb), qlb)$
- $R(b, qb)$
- $R(cons(qa, qlab1), qlab1)$
- $R(cons(qa, qlb1), qlab1)$
- $R(cons(qa, q1), q1)$
- $R(0, q1)$
- $R(cons(qb, qlab1), qlab1)$
- $R(cons(qb, q1), qlb1)$
- $R(cons(qb, q1), q1)$
- $R(b, qb)$
- $R(app(0, x), x)$
- $R(app(cons(x, y), z), cons(x, app(y, z)))$
- $R(rev(0), 0)$
- $R(rev(cons(x, y)), app(rev(y), cons(x, 0)))$
- $(R(x, y) \wedge R(y, z)) \to R(x, z)$
- $R(x, x)$
- $R(x, y) \to R(rev(x), rev(y))$

- $R(x, y) \to R(cons(z, x), cons(z, y))$
- $R(x, y) \to R(cons(x, z), cons(y, z))$
- $R(x, y) \to R(app(z, x), app(z, y))$
- $R(x, y) \to R(app(x, z), app(y, z))$

The formula $\psi_P : \exists x \exists y((R(rev(x), qrev) \wedge R(y, qlab1)) \wedge R(rev(x), y)$ expresses the negation of the correctness condition.

For this standard encoding Mace4 has failed to find a countermodel for $\Phi_P \to \psi_P$ within 250000s. However after removing the congruence axiom $R(x, y) \to R(rev(x), rev(y))$ Mace4 has found the model of size 3 (cardinality of the domain) in 0.06s. (see further details in [18]. The absence of such a congruence axiom means that no rewriting of proper subterms of $rev(\dots)$ is allowed. One can either easily argue that in TRS given above no such rewriting possible anyway, or, remaining in a pure automated verification scenario, just accept verification modulo restrictions on the rewriting strategy. Alternatively one can apply an approximation by symmetric closure, in which case Mace4 finds the model of size 3 in 0.1s.

Notice that the verification of the same system in [12] has been done using tree automata completion technique, which required interactive approximation.

7 Related work and further directions

As mentioned Section 1 the approach to verification using the modeling of protocol executions by first-order derivations and together with countermodel finding for disproving was introduced within the research on the formal analysis of cryptographic protocols [28, 27, 13, 16, 14].

In [27], apparently for the first time, explicit building of finite countermodels has been proposed as a tool to establish correctness of cryptographic protocols. It has been illustrated by an example, where a countermodel was produced manually, and the automation of the process has not been discussed. The later work [13] has shown how a countermodel produced during the verification of cryptographic protocols can be converted into a formal induction proof.

Later FCM approach has been applied to the various problems of infinite-state and parameterized system [18, 19, 20, 21, 23]. In [20] it was shown that FCM provides a decision procedure for the verification of safety properties of lossy channel systems and applications to the verification of parameterized cache coherence protocols are presented . In [21, 23] the relative completeness of FCM with respect to regular model checking and regular tree model checking, respectively, has been established. Despite its simplicity, in many cases FCM has turned out to be very efficient and comparable with or even outperforming (e.g. in [23]) the best reported alternative methods.

From the viewpoint of this paper the verification problems considered in the above papers can be seen as safety problems for the specific classes of (conditional) TRS modulo equational theories.

The verification of safety properties for general term-rewriting systems using tree automata completion techniques has been addressed in [12, 9, 11]. The paper [10] presents a method based on encoding both term-rewriting system and tree automata into Horn logic and application of the static analysis techniques. The main conceptual difference between these approaches and FCM presented in this paper, is that in [12, 9, 11, 10] the safety verification is performed in two stages: first, a tree automaton approximating all reachable terms is obtained and it depends only on TRS but not on the safety property, and, second, an intersection of the language of this automaton with the language of unsafe states is computed.

FCM method we presented here operates in one stage and computing regular approximations or invariants (in terms of finite countermodels) is done for concrete safety properties. It has its disadvantage that the results of the verification of a TRS can not be re-used for the verification of different safety properties for the same TRS. On the other hand this disadvantage is compensated by a higher degree of automation and higher explanatory power of FCM method as our experimental results suggest. Furthermore, as the Example 6.1 has demonstrated the invariants for specific safety properties might be much simpler than the approximations of all reachable terms. That raises hope that the scalability, which is a potential issue for FCM applications, can be dealt with. Tree automata completion has shown that it scales well on very large TRS and large tree automata (with thousands of rewrite rules and automata states) [3]. The applicability of FCM for the verification of specific safety properties of the large systems requires further investigation.

Another feature of FCM approach which makes it different from tree automata completion techniques is that it does not depend on linearity of TRS.

Perhaps, conceptually closest work to that reported in this paper has appeared very recently in [4]. In the context of safety verification the computation of the overapproximation of the set of reachable terms is reduced in [4] to the problem of instantiation of a symbolic tree automaton, which in turn is reduced to the satisfiability testing of a boolean combination of equalities and inequalities between variables. The latter task is then delegated to the second-order satisfiability checker Mona. Similar to FCM and unlike to the methods reported in [12, 9, 11, 10] approximations of reachable terms are computed there with respect to a particular safety property. Theoretical and practical comparisons of FCM and the method of [4] is a very interesting topic for future work.

Finally, we would like to notice that the translations we have defined in this paper can be seen as the axiomatizations of rewriting theories in a sense of rewriting logic [25] within the standard first-order logic, where binary predicate R represents the entailment $\vdash$ for rewriting theory. Rewriting logic formalizing conditional and modulo theory rewriting may serve as an unifying formalism for representing all applications of FCM. Development of such an approach is another direction for future work.

References

1 Abdulla, P.A., Jonsson,B., Nilsson, M., & Saksena, M., (2004) A Survey of Regular Model Checking, In Proc. of CONCUR'04, volume 3170 of LNCS, pp 35–58, 2004.

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

3 Boichut, Y., Genet, T., Jensen T.P., Le Roux, L., (2007) Rewriting Approximations for Fast Prototyping of Static Analyzers, in Proc. of RTA'2007, 48–62, 2007.

4 Boichut, Y., Dao, T.-B.-H., Murat, V., (2011) Characterizing Conclusive Approximations by Logical Formulae, in Proc. of Reachability Problems 2011, pp72-84, 2011

5 Caferra, R., Leitsch, A., & Peltier, M., (2004) *Automated Model Building*, Applied Logic Series, 31, Kluwer, 2004.

6 Clavel, M., Duran, F., Eker, S., Lincoln, P., Marti-Oliet, N., Meseguer, J., Talcott., C.L., 2007. All About Maude, A High-Performance Logical Framework. Vol. 4350 of Lecture Notes in Computer Science. Springer.

7 Delzanno, G., (2003) Constraint-based Verification of Parametrized Cache Coherence Protocols. *Formal Methods in System Design*, 23(3):257–301, 2003.

8 Enderton, H., (1972) *A Mathematical Introduction to Logic*, Academic Press, 1972

9 Feuillade, G., Genet, T., Tong, V.V.T.: Reachability Analysis over Term Rewriting Systems. J. Autom. Reasoning 33(3-4), 341–383 (2004).

10 Gallagher, John P., & Rosendahl, M.,(2008) Approximating Term Rewriting Systems: A Horn Clause Specification and Its Implementation. I.Cervesato, H. Veith, and A. Voronkov (Eds.): LPAR 2008, LNCS 5330, pp. 682–696, 2008.

11 Genet, T., Rusu, V., Equational Approximations for Tree Automata Completion, Journal of Symbolic Computation Volume 45, Issue 5, May 2010, Pages 574-597

12 Genet, T., Tong, V.V.T.: Reachability Analysis of Term Rewriting Systems with *timbuk*. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001, LNCS, vol. 2250, pp. 695–706. Springer, Heidelberg, 2001.

13 Goubault-Larrecq, J., (2008), Towards producing formally checkable security proofs, automatically. In: Computer Security Foundations (CSF), pp. 224–238 (2008)

14 Guttman, J., (2009) Security Theorems via Model Theory, Proceedings 16th International Workshop on Expressiveness in Concurrency, EXPRESS, EPTCS, vol. 8 (2009)

15 Habermehl, P., & Vojnar, T., (2005), Regular Model Checking Using Inference of Regular Languages, Electronic Notes in Theoretical Computer Science (ENTCS), Volume 138, Issue 3 (December 2005), pp 21–36, 2005

16 Jurjens, J., & Weber, T., (2009), Finite Models in FOL-Based Crypto-Protocol Verification, P. Degano and L. Vigan‘o (Eds.): ARSPA-WITS 2009, LNCS 5511, pp. 155–172, 2009.

17 Kusakari, K., & Chiba, Y., (2007), Higher-Order Knuth-Bendix Procedure and Its Applications, IEICE Transactions on Information and Systems, Vol.E90-D, No.4, pp.707-715, Apr 2007.

18 Lisitsa, A., (2009a), Verfication via countermodel finding `http://www.csc.liv.ac.uk/~alexei/countermodel/`

19 Lisitsa, A., (2009b), Reachability as deducibility, finite countermodels and verification. In preProceedings of AVOCS 2009, Technical Report of Computer Science, Swansea University, CSR-2-2009, pp 241-243.

20 Lisitsa, A., (2010b), Reachability as deducibility, finite countermodels and verification. In Proceedings of ATVA 2010, LNCS 6252, 233–244

21 Lisitsa, A., (2010c), Finite model finding for parameterized verification, CoRR abs/1011.0447: (2010)

22 Lisitsa, A., (2011a), First-order finite satisfiability vs tree automata in safety verification Corr abs/1107.0349:(2011)

23 Lisitsa, A.,(2011), Finite countermodels for safety verification of parameterized tree systems, CoRR abs/1107.5142:(2011)

24 McCune, W., Prover9 and Mace4 `http://www.cs.unm.edu/~mccune/mace4/`

25 Meseguer, J., (1992), Conditional Rewriting Logic as a unified model of concurrency, *Theoretical Computer Science*, 96:73–155, 1992.

26 Nilsson, M., (2005) Regular Model Checking. Acta Universitatis Upsaliensis. Uppsala Dissertations from the Faculty of Science and Technology 60. 149 pp. Uppsala. ISBN 91-554-6137-9, 2005.

27 Selinger, P., (2001), Models for an adversary-centric protocol logic. Electr. Notes Theor. Comput. Sci. 55(1) (2001)

28 Weidenbach, C., (1999), Towards an Automatic Analysis of Security Protocols in First-Order Logic, in H. Ganzinger (Ed.): CADE-16, LNAI 1632, pp. 314–328, 1999.

Triangulation in Rewriting

Vincent van Oostrom[1] and Hans Zantema[2]

1 **Department of Philosophy, Utrecht University, The Netherlands**
`Vincent.vanOostrom@phil.uu.nl`

2 **Department of Computer Science, TU Eindhoven, The Netherlands**
Institute for Computing and Information Sciences, Radboud University Nijmegen, The Netherlands
`h.zantema@tue.nl`

Abstract

We introduce a process, dubbed *triangulation*, turning any rewrite relation into a confluent one. It is more direct than usual completion, in the sense that objects connected by a peak are directly related rather than their normal forms. We investigate conditions under which this process preserves desirable properties such as termination.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases triangulation, codeterminism, completion, (co)confluence, (co)termination

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.240

Category Regular Research Paper

1 Introduction

We study the problem of deciding whether two objects are equivalent with respect to the equivalence relation generated by some rewrite relation. We do this in a fully abstract setting, that is, any binary relation on any set of objects may serve as a rewrite relation. The standard idea in such a setting is to compute the normal forms of both objects with respect to the rewrite relation, subsequently comparing whether these normal forms are equal or not. If and only if the normal forms are equal the original objects are deemed to be equivalent.

For the above to work, i.e. for it to yield a sound and complete decision procedure for equivalence, normal forms should *exist* and be *unique* within equivalence classes. For a given rewrite relation neither needs to be the case. On the one hand, the objects b and c are distinct normal forms with respect to the rewrite relation $b \leftarrowtriangle a \rightarrowtriangle c$ despite them belonging to the same equivalence class $\{a, b, c\}$ generated by $\rightarrowtriangle$ (see Figure 1 left); normal forms are *not unique* so the procedure is not sound. On the other hand, the object a does not have a normal form in the rewrite relation $a \rightarrowtriangle a$; normal forms need *not exist*, even if they are unique when they exist, hence the procedure need not be complete.

First we introduce in Section 2 a process called *triangulation* to stepwise extend an arbitrary rewrite relation in such a way that normal forms are *unique* within each equivalence class, without altering the generated equivalence relation. That is, after triangulation the decision procedure is sound: if two objects both rewrite to a normal form, then they are equivalent if and only if these normal forms are equal. For instance, triangulation would extend the above rewrite relation $b \leftarrowtriangle a \rightarrowtriangle c$ by either of the steps $b \mathrel{-\!\!\blacktriangleright} c$ (see Figure 1 right) or $b \mathrel{\blacktriangleleft\!\!-} c$, forming a triangle, hence our naming. The idea of triangulation is similar

$$\begin{tikzcd}
 & a \arrow[dl] \arrow[dr] & & & & a \arrow[dl] \arrow[dr] & \\
b & & c & & b \arrow[rr] & & c
\end{tikzcd}$$

Figure 1 $b \leftarrowtriangle a \rightarrowtriangle c$ before (left) and after (right) triangulation.

to that of abstract Knuth–Bendix completion, see e.g. [2]; the main difference is that by triangulation two not directly related objects that are related by a peak will be related directly, while in completion their normal forms will be related.

Next, in Section 3, we provide two sufficient conditions for triangulation to preserve termination, guaranteeing *existence* of normal forms when starting out with a terminating rewrite relation, thus giving rise to a sound and complete decision procedure for the generated equivalence. The first condition is a *compatibility* condition, requiring the union of the original rewrite relation and the relation used to relate the adjoined steps to be terminating. The second condition requires both these relations to be terminating on their own, and moreover the original rewrite relation to be *codeterministic*; at most one object rewrites to any given object.

Finally, in Section 4, we reflect on how the triangulation process brings about completeness. That is, looking back from the result of the completion process, i.e. from the original rewrite relation and the relation by which it is extended, we provide sufficient conditions on these two relations for their union to be complete. These conditions are abstract in that they do not require the latter being stepwise generated from the former. We also provide sufficient conditions for the result of triangulation to be cocomplete, i.e. coconfluent and coterminating.

The origin of this research was in a question of Jan Friso Groote, relevant for the typing mechanism in the tool-set of the specification language mCRL2 [5]. In a part of that mechanism types $a, b, c, d, \ldots$ are specified, some of which are equated, expressed by definitions of the shape $x \leftarrowtriangle y$ with x the definiendum and y the definiens. A natural question that came up is whether two types are equivalent in the sense that the one can be reached by the other by means of a series of definition foldings and unfoldings. For instance, in the system having six types a, b, c, d, e, f with four definitions

$$a \leftarrowtriangle b,\ b \leftarrowtriangle c,\ d \leftarrowtriangle c,\ f \leftarrowtriangle e$$

the types a and d are seen to be equivalent since they can be connected by the chain of definition foldings and unfoldings $a \leftarrowtriangle b \leftarrowtriangle c \rightarrowtriangle d$. Triangulation arose here as a method to answer the equivalence question, as it constructs unique representatives of equivalence classes, hence checking equivalence of two types reduces to checking equality of their respective representatives ($\rightarrow$-normal forms). Executing it on this example, first the peak $b \leftarrowtriangle c \rightarrowtriangle d$ is turned into a triangle by adjoining a definition $b \mathrel{-\!\!\blacktriangleright} d$, the direction being determined by some given order on the objects, here the alphabetic order. This then gives rise to a new peak $a \leftarrowtriangle b \mathrel{-\!\!\blacktriangleright} d$ which in turn is made into a triangle by adjoining $a \mathrel{-\!\!\blacktriangleright} d$, after which d, f have become the unique representatives of their respective equivalence classes. The types a and b are $\rightarrowtriangle$-equivalent since d is their common $\rightarrow$-normal form. The example also motivates our interest in studying rewrite relations that are codeterministic, as codeterminism of a system of definitions $\rightarrowtriangle$ corresponds to the natural requirement, satisfied by the example, that types are not defined twice.

2 Triangulation

We introduce triangulation as a process to turn an arbitrary rewrite relation into a confluent one, without altering the generated equivalence relation. We first quickly recapitulate the few basic notions on rewrite relations needed for this, referring the reader to [1, 6] for background information. Throughout, we will use *relation* to mean a binary relation. A relation R is said to have property *co-P* if its converse has property P.[1] A *rewrite* relation is a binary relation on a set of objects, that is, a relation having the same domain and codomain. Rewrite relations may denote notions of computation steps, and we will use arrow-like notations like $\to$, $\mathrel{-\!\!\rhd}$, $\mathrel{-\!\!\blacktriangleright}$ to denote them. For a rewrite relation $\to$, we inductively define an object a to be *terminating*, if for all objects b such that $a \to b$, b is terminating. The rewrite relation $\to$ is *terminating* if all its objects are. For a rewrite relation denoted by an arrow-like notation $\to$, its converse is denoted by the converse $\leftarrow$ of the notation. We denote the union of two rewrite relations by the union of their notations, e.g. $\leftrightarrow$ denotes $\leftarrow \cup \to$, the symmetric closure of $\to$. We say $\to$ is *total* if $a \leftrightarrow b$ for all objects a, b. We use $\mathrel{-\!\!\rhd} \cdot \mathrel{-\!\!\blacktriangleright}$ to denote the composition of $\mathrel{-\!\!\rhd}$ and $\mathrel{-\!\!\blacktriangleright}$, use $\to^n$ to denote the n-fold composition of $\to$ with itself, and $\to^=$ and $\to^+$ to denote respectively the reflexive and transitive closure of $\to$. To denote the reflexive–transitive closure of $\to$, i.e. its 'repetition', we employ the 'repetition' $\twoheadrightarrow$ of its notation, or, if clutter would arise from repeating the notation, $\to^*$. If $a \twoheadrightarrow b$ then we say that a *reduces* or *rewrites* to b. We define $\to$*-expansion* as $\to$-coreduction, i.e. $\leftarrow$-reduction, and $\to$*-convertibility* as $\leftrightarrow^*$ which is easily shown to be the equivalence closure of $\to$. A rewrite relation $\to$ is *confluent* if $\twoheadleftarrow \cdot \twoheadrightarrow \subseteq \twoheadrightarrow \cdot \twoheadleftarrow$. A rewrite relation is *complete* if it is both terminating and confluent. Further notions and notations will be introduced on a by-need basis.

Now we are ready for our main definition: we want to extend an arbitrary rewrite relation $\mathrel{-\!\!\rhd}$ to achieve desired properties like confluence. In doing so, we use a total relation R, typically a total order, to order the added new pairs.

▶ **Definition 2.1** (Triangulation)**.** Let R be a total relation on the objects of a rewrite relation $\mathrel{-\!\!\rhd}$. The *triangulation* $\mathrm{Tr}(\mathrel{-\!\!\rhd})$ of $\mathrel{-\!\!\rhd}$ with respect to R is the rewrite relation $\to = \bigcup_{n \geq 1} \to_n$, where $\to_n$ is defined inductively as follows:

- $\to_1 = \mathrel{-\!\!\rhd}$; and
- $a \to_n b$ holds for $n > 1$ if and only if
  - $a \mathrel{R} b$ and for some $m, k \geq 1$ with $m + k = n$ there is a c such that $a \leftarrow_m c \to_k b$; and
  - for no $k < n$, $a \leftrightarrow_k^= b$ holds.

Below we will assume the relation R to be total.

▶ **Example 2.2.** Triangulating the rewrite relation $\mathrel{-\!\!\rhd}$ displayed on the left in Figure 2 with respect to the usual alphabetic order on its objects $\{a, \ldots, h\}$ gives rise to the rewrite relation displayed on the right of that figure. The original arrows are labelled by 1. Next for every peak $x \leftarrow \cdot \to y$ with $x \neq y$ for which there is not yet an arrow between x and y an arrow $x \to_2 y$ is created or conversely, depending on whether $x \mathrel{R} y$ or $y \mathrel{R} x$ holds. This is continued until after creating the $\to_5$ arrow, there is a direct arrow between x and y for all peaks $x \leftarrow \cdot \to y$ with $x \neq y$.

▶ Remark. The index n of a step $a \to_n b$ is the number of $\mathrel{-\!\!\rhd}$-steps in a proof showing that a and b are $\mathrel{-\!\!\rhd}$-convertible. Counting each axiom ($\mathrel{-\!\!\rhd}$-step) and transitivity rule employed

[1] We will in particular be interested in codeterminism, coconfluence, cotermination and cocompleteness.

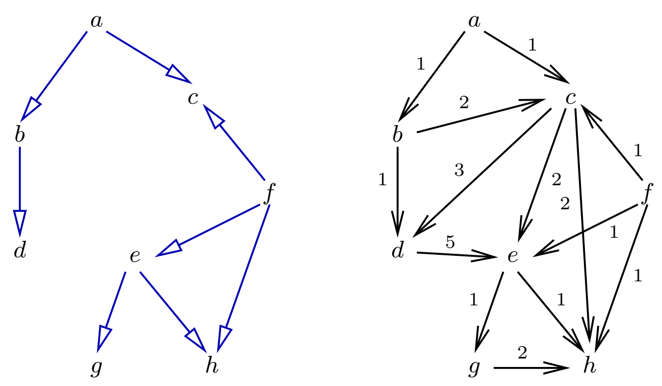


Figure 2 A rewrite relation $\mathrel{-\!\triangleright}$ (left) and its triangulation (right).

in such a proof, this amounts to the same thing as half of (one plus the size of that proof). In Example 2.2 the step $d \to_5 e$ witnesses a proof of size 9 ($= 2 \times 5 - 1$) based on the $\mathrel{-\!\triangleright}$-conversion $d \mathrel{\triangleleft\!-} b \mathrel{\triangleleft\!-} a \mathrel{-\!\triangleright} c \mathrel{\triangleleft\!-} f \mathrel{-\!\triangleright} e$ consisting of 5 steps and 4 applications of the transitivity rule as witnessed by first $b \to_2 c$ and $c \to_2 e$, then $d \leftarrow_3 c$, and finally $d \to_5 e$.

The rewrite relation in the example is finite, hence triangulation must stop in the sense that no new steps will be adjoined from some stage on (in fact, from stage 5 on). But also for infinite rewrite relations the process may well stop. Triangulation always results in what we will call an *affluent* rewrite relation without altering the generated equivalence relation.

▶ **Definition 2.3** (Affluence). A pair $\mathrel{-\!\triangleright}, \mathrel{-\!\blacktriangleright}$ of rewrite relations is *affluent*[2] if $\mathrel{\triangleleft\!\!\triangleleft\!-} \cdot \mathrel{-\!\blacktriangleright\!\!\blacktriangleright} \subseteq \mathrel{\triangleleft\!\!\triangleleft\!-} \cup \mathrel{-\!\blacktriangleright\!\!\blacktriangleright}$, *one-step* affluent if $\mathrel{\triangleleft\!-} \cdot \mathrel{-\!\blacktriangleright} \subseteq \mathrel{\triangleleft\!-} \cup \mathrel{-\!\blacktriangleright}$, and *locally* affluent if $\mathrel{\triangleleft\!-} \cdot \mathrel{-\!\blacktriangleright} \subseteq \mathrel{\triangleleft\!\!\triangleleft\!-} \cup \mathrel{-\!\blacktriangleright\!\!\blacktriangleright}$. These notions pertain to a single rewrite relation $\to$ via the pair $\to, \to$.

Observe that in term rewriting affluence seldomly occurs: even with respect to the single rule $a \to b$, affluence does not hold for the term $f(a, a)$. So both affluence and triangulation are about rewrite relations in abstract settings, typically not described by a term rewriting system.

▶ Remark. One-step affluence of $\mathrel{-\!\triangleright}^=, \mathrel{-\!\blacktriangleright}^=$ is equivalent to $\mathrel{\triangleleft\!-} \cdot \mathrel{-\!\blacktriangleright} \subseteq (\mathrel{\triangleleft\!-} \cup \mathrel{-\!\blacktriangleright})^=$, hence implies sub-commutation $\mathrel{\triangleleft\!-} \cdot \mathrel{-\!\blacktriangleright} \subseteq \mathrel{-\!\blacktriangleright}^= \cdot \mathrel{\triangleleft\!-}^=$. In turn, local affluence of $\mathrel{-\!\triangleright}, \mathrel{-\!\blacktriangleright}$ implies local commutation $\mathrel{\triangleleft\!-} \cdot \mathrel{-\!\blacktriangleright} \subseteq \mathrel{-\!\blacktriangleright\!\!\blacktriangleright} \cdot \mathrel{\triangleleft\!\!\triangleleft\!-}$ and affluence of $\mathrel{-\!\triangleright}, \mathrel{-\!\blacktriangleright}$ implies commutation $\mathrel{\triangleleft\!\!\triangleleft\!-} \cdot \mathrel{-\!\blacktriangleright\!\!\blacktriangleright} \subseteq \mathrel{-\!\blacktriangleright\!\!\blacktriangleright} \cdot \mathrel{\triangleleft\!\!\triangleleft\!-}$. In case of a single rewrite relation, the same holds, replacing commutation by confluence.

The standard example showing that local commutation does not imply commutation, $b \mathrel{\triangleleft\!-} a \mathrel{\substack{-\!\blacktriangleright \\ \triangleleft\!-}} a' \mathrel{-\!\blacktriangleright} c$, shows local affluence does not imply affluence. Analogous to the fact that one-step commutation implies commutation by the former being preserved under taking reflexive and transitive closures, one-step affluence implies affluence:

▶ **Lemma 2.4.** *If $\mathrel{-\!\triangleright}, \mathrel{-\!\blacktriangleright}$ is one-step affluent, then so are $\mathrel{-\!\triangleright}^=, \mathrel{-\!\blacktriangleright}^=$ and $\mathrel{-\!\triangleright}^+, \mathrel{-\!\blacktriangleright}^+$.*

Proof. Suppose the pair $\mathrel{-\!\triangleright}, \mathrel{-\!\blacktriangleright}$ is one-step affluent.

That $\mathrel{-\!\triangleright}^=, \mathrel{-\!\blacktriangleright}^=$ is one-step affluent follows from $\mathrel{\triangleleft\!-}^= \cdot \mathrel{-\!\blacktriangleright}^= = \mathsf{id} \cup \mathrel{\triangleleft\!-} \cup \mathrel{-\!\blacktriangleright} \cup (\mathrel{\triangleleft\!-} \cdot \mathrel{-\!\blacktriangleright})$.

[2] The idea is that whereas the standard notion of *confluence* expresses that rewrite sequences (viewed as streams) may 'flow together', we use *affluence* in its original (archaic) sense to express that one of them may 'flow to' (is a tributary of) the other.

To prove that $\rightarrowtriangle^+$, $\mathrel{-\!\!\blacktriangleright}^+$ is one-step affluent, we show for $n,m \geq 0$, $a \leftarrowtriangle^n \cdot \leftarrowtriangle \cdot \mathrel{-\!\!\blacktriangleright} \cdot \mathrel{-\!\!\blacktriangleright}^m$ b implies a $(\leftarrowtriangle^+ \cup \mathrel{-\!\!\blacktriangleright}^+)$ b, by induction on $n+m$. By assumption $a \leftarrowtriangle^n \cdot (\leftarrowtriangle \cup \mathrel{-\!\!\blacktriangleright}) \cdot \mathrel{-\!\!\blacktriangleright}^m$ b. Suppose w.l.o.g. $a \leftarrowtriangle^n \cdot \leftarrowtriangle \cdot \mathrel{-\!\!\blacktriangleright}^m$ b. Then we conclude to $a \leftarrowtriangle^+ b$ if $m = 0$. Otherwise we conclude to a $(\leftarrowtriangle^+ \cup \mathrel{-\!\!\blacktriangleright}^+)$ b by the induction hypothesis. ◀

▶ **Theorem 2.5.** *Let $\to$ = $Tr(\rightarrowtriangle)$ be the triangulation of any rewrite relation $\rightarrowtriangle$. Then $\to$ is affluent and $\leftrightarrow^* = \leftrightarrowtriangle^*$.*

Proof. To prove affluence of $\rightarrowtriangle$ it suffices by Lemma 2.4 and the remark above it, to prove $\leftarrow \cdot \to \subseteq \leftrightarrow^=$. To that end, we show that for all natural numbers $n,m \geq 1$, $a \leftarrow_n c \to_m b$ implies $a \leftrightarrow^= b$. By the triangulation construction then either a is $\bigcup_{1 \leq k < n+m} \leftrightarrow_k^=$-related to b and we are done, or a is not so related to b and then $a \leftrightarrow_{n+m} b$ by the assumed totality of the relation R.

To prove $\leftrightarrow^* = \leftrightarrowtriangle^*$ it suffices, since $\to = \bigcup_{n \geq 1} \to_n$, to show $\to_n \subseteq \leftrightarrowtriangle^*$ for all $n \geq 1$ by induction on n. In the base case $\to_1 = \rightarrowtriangle \subseteq \leftrightarrowtriangle^*$. If $a \to_{n+m} b$ because $a \leftarrow_n c \to_m b$ for some object c and natural numbers n,m, then by the induction hypothesis $a \leftrightarrowtriangle^* c \leftrightarrowtriangle^* b$, and we conclude by transitivity of $\leftrightarrowtriangle^*$. ◀

▶ **Corollary 2.6.** *The triangulation $Tr(\rightarrowtriangle)$ is confluent for every rewrite relation $\rightarrowtriangle$.*

Having established this basic result, we investigate in the next section on which rewrite relations triangulation is a completion process, i.e. for which rewrite relation does triangulation preserve termination?

3 Completion

When does triangulation yield a *complete* rewrite relation? That is, when does triangulation yield a rewrite relation that is both confluent and terminating? We first present an example showing that, in general, triangulation fails to do so. Analysing the example, we then propose two sufficient conditions for triangulation to preserve termination of the rewrite relation, i.e. for triangulation to be a completion process.

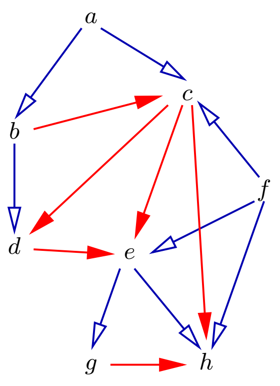


Figure 3 Rendering of Figure 2 using convention to decompose $\to$ into $\rightarrowtriangle$, $\mathrel{-\!\!\blacktriangleright}$.

To ease discussing the various examples and conditions, we will from now on, when discussing triangulation, use $\mathrel{-\!\!\blacktriangleright}$ to denote $\bigcup_{n>1} \to_n$. Hence $\to = \rightarrowtriangle \cup \mathrel{-\!\!\blacktriangleright}$ and this union is disjoint; the triangulation $\to$ consists of the original relation $\rightarrowtriangle$ and the *triangulating* relation $\mathrel{-\!\!\blacktriangleright}$, cf. Figure 3.

▶ **Example 3.1.** Consider the rewrite relation $\mathrel{-\!\!\triangleright}$ and its triangulation with respect to the usual greater-than relation $>$ as displayed in Figure 4. The resulting rewrite relation $\to \,=\, \mathrel{-\!\!\triangleright} \cup \mathrel{-\!\!\blacktriangleright}$ is not terminating; it is even cyclic: $0 \mathrel{-\!\!\triangleright} 4 \mathrel{-\!\!\blacktriangleright} 3 \mathrel{-\!\!\triangleright} 1 \mathrel{-\!\!\blacktriangleright} 0$.

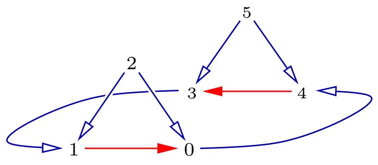


Figure 4 Failure of triangulation to yield a complete rewrite relation.

What causes that triangulation succeeds in yielding a complete rewrite relation in Example 2.2? Below we present two conditions, compatibility and codeterminism, each being sufficient for triangulation to be a completion process.

3.1 Compatibility

Our first (trivial) condition is based on the observation that despite that both the original rewrite relation $\mathrel{-\!\!\triangleright}$ and the greater-than relation $>$ with respect to which triangulation takes place in Example 3.1 are terminating, they are not *compatible* in the sense that their union is not terminating; it even is cyclic, e.g. $0 \mathrel{-\!\!\triangleright} 4 > 0$. Since the $\mathrel{-\!\!\blacktriangleright}$-steps adjoined by triangulation conform to the relation with respect to which triangulation takes place, the latter being compatible with $\mathrel{-\!\!\triangleright}$ guarantees that termination is preserved:

▶ **Theorem 3.2.** *The triangulation $Tr(\mathrel{-\!\!\triangleright})$ of $\mathrel{-\!\!\triangleright}$ with respect to R is terminating if $\mathrel{-\!\!\triangleright} \cup R$ is terminating.*

Proof. Combining $\to_1 \,=\, \mathrel{-\!\!\triangleright}$ with $\to_n \subseteq R$, for all $n > 1$, we conclude to $\to \,=\, \bigcup_{n \geq 1} \to_n \subseteq \mathrel{-\!\!\triangleright} \cup R$, hence to termination of $\to$. ◀

▶ Remark. The termination assumption on $\mathrel{-\!\!\triangleright} \cup R$ could have been rephrased as: $\mathrel{-\!\!\triangleright} \subseteq R$ and R is terminating. Clearly, the latter entails the former. To see the converse, note that if $a \mathrel{-\!\!\triangleright} b$ but $a \not R\, b$, then $b \mathrel{R} a$ by totality of R, hence $a \mathrel{-\!\!\triangleright} b \mathrel{R} a$ contradicting termination of $\mathrel{-\!\!\triangleright} \cup R$. Therefore $\mathrel{-\!\!\triangleright} \subseteq R$ and R is terminating.

▶ Remark. In ordinary completion, instead of adjoining a step between b and c as in triangulation, a step between their $\mathrel{-\!\!\triangleright}$-normal forms is adjoined if they are distinct. Under the conditions of the theorem, also ordinary completion yields a complete rewrite relation. However, ordinary completion results in confluence, not the stronger affluence guaranteed by triangulation.

▶ Remark. If triangulation results in a complete rewrite relation, then, although *unboundedly* many stages may have been needed, there cannot be *unbounded creation*. More precisely, there is no infinite sequence of triangulating steps such that each step in the sequence is a cause, one of the two (see Figure 5 left), in the triangulation process for the next step in the sequence, as unbounded creation would contradict termination of the obtained rewrite relation (see Figure 5 right).

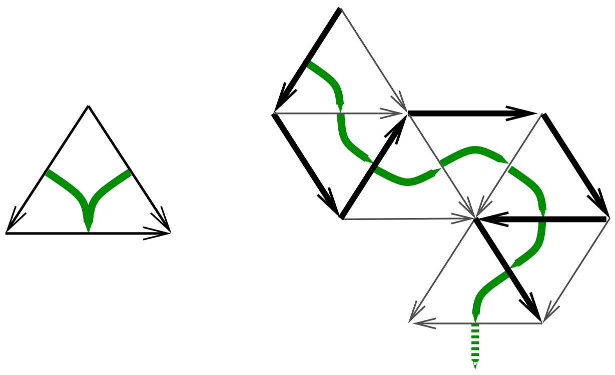


Figure 5 How steps in a peak cause its triangulating step (left) and how an infinite causal chain would give rise to an infinite sequence of steps through the sources of the triangulating steps (right).

3.2 Codeterminism

Our second (nontrivial) condition is based on the observation that the original rewrite relation $\rightarrowtriangle$ in Example 3.1 is not *codeterministic*: there are objects that are the target of more than one rewrite step, viz. $0 \rightarrowtriangle 4 \leftarrowtriangle 5$.

▶ **Definition 3.3.** A binary relation R is *deterministic* if for all objects a in its domain, and all objects b,c in its codomain, $a\ R\ b$ and $a\ R\ c$ imply $b = c$.

Forbidding the above configurations is captured by requiring the rewrite relation to be *codeterministic* per our earlier convention of a relation having a property co-P if its converse has property P.

$$\begin{tikzcd}
a \arrow[r] \arrow[d] & b \arrow[r] & c \arrow[r] \arrow[d] & d \arrow[r] & e \arrow[llll, bend right] \\
f \arrow[r] \arrow[d] & i \arrow[d] & h & & \\
g & j & & &
\end{tikzcd}$$

Figure 6 Codeterministic rewrite relation.

▶ **Example 3.4.** Neither of the rewrite relations of Example 2.2 (Figure 2 left) and Example 3.1 is codeterministic. For the latter this was observed above. The former fails to be codeterministic because, e.g., $a \rightarrowtriangle c \leftarrowtriangle f$. Both the system of definitions given in the introduction in Figure 1 and the rewrite relation displayed in Figure 6 are codeterministic.

Note that the graph in Figure 6 consists of a number of trees branching off from a cycle ($a \rightarrowtriangle b \rightarrowtriangle c \rightarrowtriangle d \rightarrowtriangle e \rightarrowtriangle a$). It is easy to see (branching 'off' is allowed by codeterminism, but branching 'in' is not) that this holds in general: each component of the graph of a codeterministic rewrite relation consists of a number (possibly 1) of trees branching off (if at all) from pairwise distinct objects lying at a (possibly empty) cycle. Thus the graph of an *acyclic* codeterministic rewrite relation is a forest of trees.[3]

[3] The trees may be infinite: both infinitely branching and non-rooted trees are allowed.

▶ Remark. Although it is not unreasonable to require acyclicity, in any case for systems of definitions as in the introduction, one can well imagine an infinite setting without maximal types, that is, for every type there's a definition allowing it to be folded further. In Section 4 we also provide a dual approach, choosing maximally *unfolded* representatives. For the moment we will view termination simply as necessary for the approach via maximally *folded* representatives to make sense.

Intuitively, triangulating is a completion process for codeterministic rewrite relations, since despite that triangulating the different branches from a node in a forest will create new steps 'spanning the gaps' between these branches, no cycles will be created nor will disjoint trees be joined by triangulating, hence termination will be preserved. As formalising this intuition exactly turned out to be tedious,[4] we instead provide a short proof based on the following result due to Doornbos and von Karger. Variations on this result are given by Dershowitz in [3].

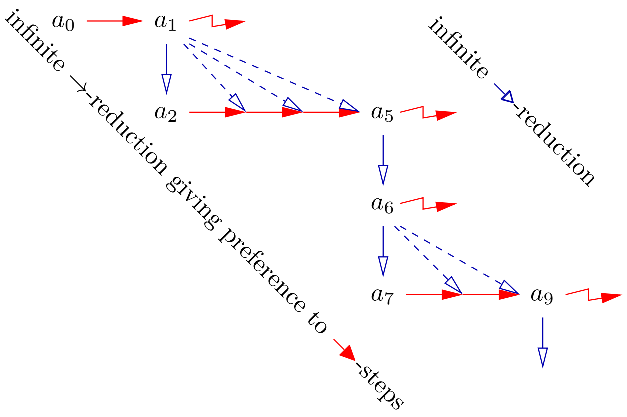

a_0
a_1
a_2
a_5
a_6
a_7
a_9
infinite $\to$-reduction giving preference to $-\!\!\blacktriangleright$-steps
infinite $-\!\!\triangleright$-reduction

Figure 7 Ramsey-style construction in proof of Lemma 3.5.

▶ **Lemma 3.5** ([4]). *For rewrite relations $-\!\!\triangleright$, $-\!\!\blacktriangleright$ let $\to = -\!\!\triangleright \cup -\!\!\blacktriangleright$. Then $\to$ is terminating if $-\!\!\triangleright$ and $-\!\!\blacktriangleright$ are terminating, and $-\!\!\triangleright \cdot -\!\!\blacktriangleright \subseteq -\!\!\triangleright \cup (-\!\!\blacktriangleright \cdot \twoheadrightarrow)$.*[5]

Proof. We provide a Ramsey-style proof, see Figure 7, as an alternative to the calculational proof in [4]. Without loss of generality we may assume $-\!\!\triangleright$ and $-\!\!\blacktriangleright$ to be disjoint (otherwise consider $-\!\!\triangleright - -\!\!\blacktriangleright$ and $-\!\!\blacktriangleright$).

For a proof by contradiction, suppose an object a_0 such that $\infty(a_0)$ were to exist, writing $\infty(a)$ to denote that a allows an infinite $\to$-reduction. Such an infinite $\to$-reduction sequence may then be constructed through objects a_n with $\infty(a_n)$, while giving preference to $-\!\!\blacktriangleright$-steps. Formally: suppose the sequence has been constructed up to a_n. If there exists an object a such that $a_n -\!\!\blacktriangleright a$ and $\infty(a)$, then we set a_{n+1} to a. Otherwise we set a_{n+1} to any object a such that $a_n -\!\!\triangleright a$ and $\infty(a)$ (which must exist since $\infty(a_n)$ and $\to = -\!\!\triangleright \cup -\!\!\blacktriangleright$).

We show one can construct an infinite $-\!\!\triangleright$-subsequence through objects *that do not allow a $-\!\!\blacktriangleright$-step to an object b with $\infty(b)$*. Formally: we start out with the empty subsequence for some object a_n for a pair of indices n,$n+1$ such that $a_n -\!\!\triangleright a_{n+1}$. Such a pair must exist since otherwise the original reduction sequence would be an infinite $-\!\!\blacktriangleright$-reduction sequence.

[4] Still, that formalisation may be interesting, e.g. from a proof-theoretic point of view.
[5] The latter property was dubbed *lazy commutation* in [3].

Per construction of the original sequence, $a_n \mathrel{-\!\triangleright} a_{n+1}$ entails that a_n does not allow a $\mathrel{-\!\blacktriangleright}$-step to an object b with $\infty(b)$. Next, suppose to have a subsequence ending in a_n for some pair of indices $n < m$ such that $a_n \mathrel{-\!\triangleright} a_m$.

- In case $a_m \mathrel{-\!\triangleright} a_{m+1}$ (e.g. $a_1 \mathrel{-\!\triangleright} a_5 \mathrel{-\!\triangleright} a_6$ for $n = 1$, $m = 5$ in Figure 7), we extend the subsequence with a_m and continue with the pair of indices m,$m + 1$.
- Otherwise $a_n \mathrel{-\!\triangleright} a_m \mathrel{-\!\blacktriangleright} a_{m+1}$ (e.g. $a_6 \mathrel{-\!\triangleright} a_7 \mathrel{-\!\blacktriangleright} a_8$ in Figure 7), hence by assumption $a_n \; (\mathrel{-\!\triangleright} \cup (\mathrel{-\!\blacktriangleright} \cdot \twoheadrightarrow)) \; a_{m+1}$. Since by construction a_n does not allow a $\mathrel{-\!\blacktriangleright}$-step to an object b with $\infty(b)$, but $\infty(a_{m+1})$, in fact $a_n \mathrel{-\!\triangleright} a_{m+1}$ must hold and we may continue with the pair of indices n,$m + 1$.

Noting that we always eventually end up in the first case (otherwise the original sequence would contain an infinite $\mathrel{-\!\blacktriangleright}$-reduction sequence, cf. $a_6 \mathrel{-\!\triangleright} a_9$ in Figure 7), the process yields an infinite $\mathrel{-\!\triangleright}$-reduction sequence (in Figure 7 the sequence $a_1, a_5, a_6, a_9, \ldots$), contradicting termination of $\mathrel{-\!\triangleright}$. ◀

▶ Remark. The proof of Lemma 3.5 in fact shows a stronger property than preservation of termination of rewrite *relations*: it shows that any *object* allowing an infinite $\to$-reduction also allows an infinite $\to$-reduction having either an infinite $\mathrel{-\!\triangleright}$-tail or an infinite $\mathrel{-\!\blacktriangleright}$-tail.

▶ **Theorem 3.6.** *Triangulating with respect to a terminating relation preserves termination of codeterministic rewrite relations.*

Proof. Let $\to$ be the triangulation $\mathrm{Tr}(\mathrel{-\!\triangleright}) = \bigcup_{n\geq 1} \to_n$ of the codeterministic terminating rewrite relation $\mathrel{-\!\triangleright}$ and let $\mathrel{-\!\blacktriangleright} \; = \bigcup_{n>1} \to_n$ be the triangulating rewrite relation, which is terminating since it is contained in the terminating relation R. Since $\to \; = \; \mathrel{-\!\triangleright} \cup \mathrel{-\!\blacktriangleright} \; \subseteq \; \mathrel{-\!\triangleright}^+ \cup \mathrel{-\!\blacktriangleright} \; \subseteq \; \to^+$, to prove termination of $\to$ it suffices by Lemma 3.5 to prove $\mathrel{-\!\triangleright}^+ \cdot \mathrel{-\!\blacktriangleright} \; \subseteq \; \mathrel{-\!\triangleright}^+ \cup (\mathrel{-\!\blacktriangleright} \cdot \twoheadrightarrow)$. This is an immediate consequence of the following claim.

Claim: $\mathrel{-\!\triangleright}^+ \cdot \leftrightarrow_n \; \subseteq \; \mathrel{-\!\triangleright}^+ \cup (\mathrel{-\!\blacktriangleright} \cdot \twoheadrightarrow)$ for all $n > 1$.

We prove this claim by induction on n, for $n > 1$. Suppose $a \mathrel{-\!\triangleright}^+ b \leftrightarrow_n c$ for some $n > 1$. We have to prove that either $a \mathrel{-\!\triangleright}^+ c$ or $a \mathrel{-\!\blacktriangleright} \cdot \twoheadrightarrow c$. By definition of triangulation $b \leftrightarrow_n c$ implies that there is an object d and there are natural numbers m, k both ≥ 1 and $< n$ such that $b \leftarrow_m d \to_k c$. We distinguish cases on whether or not $m = 1$.

- If $m = 1$, then $b \mathrel{\triangleleft\!-} d$ by definition of triangulation, hence $a \mathrel{-\!\triangleright\!\!\triangleright} d$ by codeterminism of $\mathrel{-\!\triangleright}$. If $a = d$ then we are done since then $a \mathrel{-\!\blacktriangleright} c$. Otherwise we conclude by the induction hypothesis for $a \mathrel{-\!\triangleright}^+ d \to_k c$.
- If $m > 1$, then by the induction hypothesis for $a \mathrel{-\!\triangleright}^+ b \leftarrow_m d$ we conclude that $a \; (\mathrel{-\!\triangleright}^+ \cup (\mathrel{-\!\blacktriangleright} \cdot \twoheadrightarrow)) \; d$. If $a \mathrel{-\!\triangleright}^+ d$ then we conclude by the induction hypothesis as in the previous item. Otherwise $a \mathrel{-\!\blacktriangleright} \cdot \twoheadrightarrow d \to c$. ◀

▶ **Corollary 3.7.** *Triangulating a terminating codeterministic rewrite relation with respect to a terminating relation is a completion process.*

The corollary does not necessarily yield a *decision procedure* for deciding equivalence of terminating codeterministic rewrite relations, as the triangulation process may well be infinite itself, as it is the case for the rewrite relation in Figure 8.

Acyclicity too is preserved by triangulation of codeterministic relations.

▶ **Corollary 3.8.** *Triangulating an acyclic codeterministic rewrite relation with respect to an acyclic relation, yields an acyclic relation.*

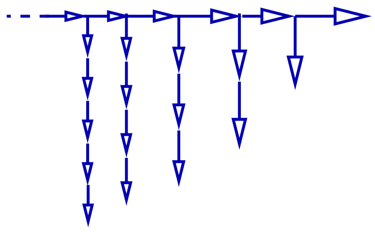


Figure 8 Rewrite relation requiring infinite triangulation.

Proof. If a cycle were to exist in the triangulation, then this cycle would be generated from a finite part of the rewrite relation. On this finite part both the rewrite relation and the total relation with respect to which it is triangulated, are terminating by acyclicity, hence by Theorem 3.6 there can be no cycle. ◀

4 Completed

The previous section provided sufficient conditions on the original rewrite relation $\rightarrowtriangle$ and the relation R with respect to which it was triangulated, for the triangulation process to yield a complete rewrite relation. In this section, we reverse the process and look for conditions on rewrite relations $\rightarrowtriangle$, $\mathrel{-\!\blacktriangleright}$ that are sufficient for their union to be complete. Although one still may think of $\rightarrowtriangle$ as the original rewrite relation and $\mathrel{-\!\blacktriangleright}$ as the triangulating rewrite relation, the conditions provided will be abstract in that they only concern $\rightarrowtriangle$ and $\mathrel{-\!\blacktriangleright}$. In particular, the conditions refer neither to the triangulation process nor to the relation R used in it; they may (and will) of course refer to the characteristic properties the triangulation process brings about. Throughout $\rightarrow$ will be used to denote $\rightarrowtriangle \cup \mathrel{-\!\blacktriangleright}$.[6]

On the one hand, triangulation guarantees every peak to give rise to a triangle:

$$\leftarrow \cdot \rightarrow \subseteq \leftrightarrow^{=} \qquad \textbf{(affluence)}$$

On the other hand, it guarantees that every $\mathrel{-\!\blacktriangleright}$-step is caused by triangulation:

$$\mathrel{-\!\blacktriangleright} \subseteq \leftarrow \cdot \rightarrow \qquad \textbf{(triangulation)}$$

As these properties are characteristic of the triangulation process of Section 2, we will assume both throughout this section.

Since **(affluence)** states one-step affluence of $\rightarrow^{=}$ which in turn implies that $\rightarrow$ is confluent (Lemma 2.4), to obtain completeness it only remains to investigate which conditions on $\rightarrowtriangle$, $\mathrel{-\!\blacktriangleright}$ are sufficient to guarantee termination of $\rightarrow$. In the case that $\rightarrowtriangle$ is not codeterministic, one cannot do much more than crudely require that $\rightarrow$ itself is terminating. Of course, one may try to employ compatibility results such as Lemma 3.5, to reduce termination of $\rightarrow$ to that of its constituting relations $\rightarrowtriangle$, $\mathrel{-\!\blacktriangleright}$, but as the following example shows, employing that lemma is tied closely to the codeterministic case.

▶ **Example 4.1.** The union of $a \leftarrowtriangle b \rightarrowtriangle c \leftarrowtriangle d \rightarrowtriangle e$ and $c \mathrel{-\!\blacktriangleright} e \mathrel{\blacktriangleleft\!-} a \mathrel{\blacktriangleleft\!-} c$ is terminating, but Lemma 3.5 cannot be used to show this as $b \rightarrowtriangle a \mathrel{-\!\blacktriangleright} e$ but neither $b \rightarrowtriangle^{+} e$ nor $b \mathrel{-\!\blacktriangleright} \cdot \twoheadrightarrow e$. Note that $\rightarrowtriangle$ is not codeterministic.

[6] Assuming $\rightarrowtriangle$ and $\mathrel{-\!\blacktriangleright}$ to be disjoint would not affect the results in this section.

Hence(forth), we restrict ourselves to the codeterministic case below. That is, we will assume the following property on top of the **(affluence)** and **(triangulation)** assumptions:

$$\mathrel{-\!\!\rhd} \cdot \mathrel{\lhd\!\!-} \subseteq \mathsf{id}; \qquad\qquad \textbf{(codeterminism)}$$

4.1 Preservation of acyclicity by finiteness

We first show that for *finite* rewrite relations, as is the case for the motivating systems of definitions in the introduction, the above three conditions are already sufficient to guarantee completeness of $\to$, if $\mathrel{-\!\!\rhd}$, $\mathrel{-\!\!\blacktriangleright}$ are assumed terminating. After that we show this does not hold for infinite rewrite relations.

Noting that for finite rewrite relations termination coincides with acyclicity, our first result can be formulated as a preservation of acyclicity result.

▶ **Theorem 4.2.** *For finite rewrite relations* $\mathrel{-\!\!\rhd}$, $\mathrel{-\!\!\blacktriangleright}$ *and* $\to \; = \; \mathrel{-\!\!\rhd} \cup \mathrel{-\!\!\blacktriangleright}$*, if properties* ***(triangulation)*** *and* ***(codeterminism)*** *hold and* $\mathrel{-\!\!\rhd}$, $\mathrel{-\!\!\blacktriangleright}$ *are acyclic, then* $\to$ *is acyclic.*

Proof. Assume $\to$ admits a cycle. With a cycle we associate the multiset of elements on it, by which cycles can be compared by a well-founded order: the multiset order induced by $\mathrel{-\!\!\blacktriangleright}$. Now we take a $\to$-cycle that is minimal with respect to this multiset order, that is, there exists no cycle that is strictly smaller with respect to the multiset order.

As $\mathrel{-\!\!\rhd}$ and $\mathrel{-\!\!\blacktriangleright}$ are acyclic, our cycle contains both $\mathrel{-\!\!\rhd}$ and $\mathrel{-\!\!\blacktriangleright}$-steps. Moreover, it also contains a $\mathrel{-\!\!\rhd}$-step followed by a $\mathrel{-\!\!\blacktriangleright}$-step:

$$a \mathrel{-\!\!\rhd} b \mathrel{-\!\!\blacktriangleright} c \twoheadrightarrow a.$$

Now we choose c_1 such that $b \leftarrow c_1 \to c$, using property **(triangulation)**. As long as $b \mathrel{\blacktriangleleft\!\!-} c_i$ for increasing i we repeat the following: choose c_{i+1} such that $b \leftarrow c_{i+1} \to c_i$, again using property **(triangulation)**. Now one of the following two cases holds:

- This process stops. Then for the last chosen c_i we have $c_i \mathrel{-\!\!\rhd} b$, see Figure 9. By property **(codeterminism)** we conclude $a = c_i$. Since $c_j \mathrel{-\!\!\blacktriangleright} b$ for all j satisfying $1 \leq j < i$, we conclude that the cycle via $\{c_j \mid 1 \leq j < i\}$

  $$a = c_i \twoheadrightarrow c \twoheadrightarrow a$$

  is smaller than our original minimal cycle: the element b has been replaced by the (possibly empty) multiset $\{c_j \mid 1 \leq j < i\}$, contradiction.

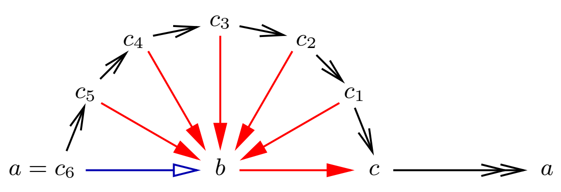


Figure 9 Smaller cycle if erecting cotriangles stops.

- This process goes on forever. Then by finiteness of the set we obtain $c_j = c_i$ for some $j > i$, yielding a new cycle $c_j \to c_{j-1} \twoheadrightarrow c_i = c_j$. Since $b \mathrel{\blacktriangleleft\!\!-} c_\kappa$ for all κ, all elements in this new cycle are less than the element b occurring in the original minimal cycle, again contradicting minimality.

As both cases contradict the assumption of the existence of a $\to$-cycle, we have proved that $\to$ is acyclic. ◀

▶ **Corollary 4.3.** *For finite rewrite relations $\rightarrowtriangle$, $\mathrel{-\!\!\blacktriangleright}$ and $\to\ =\ \rightarrowtriangle \cup \mathrel{-\!\!\blacktriangleright}$, if properties* **(affluence)***,* **(triangulation)** *and* **(codeterminism)** *hold and $\rightarrowtriangle$, $\mathrel{-\!\!\blacktriangleright}$ are terminating, then $\to$ is complete.*

Somewhat surprisingly, this result does not generalise to rewrite relations on infinite sets of objects as illustrated by the following counterexamples.

▶ Counterexamples 4.4. Take as objects the set of natural numbers and let

- $\rightarrowtriangle\ = \{(n+1, n) \mid n \geq 2\} \cup \{(2,0),(0,1)\}$; and
- $\mathrel{-\!\!\blacktriangleright}\ = \{(n,1) \mid n \geq 2\} \cup \{(1,0)\}$.

Then except for finiteness, all conditions of Corollary 4.3 are satisfied yet $\to$ is not terminating; it admits the cycle $0 \rightarrowtriangle 1 \mathrel{-\!\!\blacktriangleright} 0$ (see Figure 10 left).

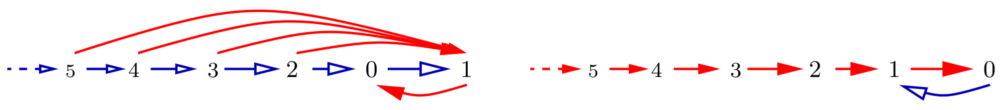


Figure 10 Loss of termination by infinite $\rightarrowtriangle$-expansion (left) and infinite $\mathrel{-\!\!\blacktriangleright}$-expansion (right).

For another example, take as objects the set of natural numbers and let

- $\rightarrowtriangle$ relate 0 to 1;
- $\mathrel{-\!\!\blacktriangleright}$ be the greater-than relation.

Again all conditions of Corollary 4.3 except for finiteness are satisfied, yet $\to$ is not terminating; it admits the cycle $0 \rightarrowtriangle 1 \mathrel{-\!\!\blacktriangleright} 0$ (see Figure 10 right where transitive $\mathrel{-\!\!\blacktriangleright}$-edges have been omitted).

In the next two sections, we investigate how to regain the preservation result for (potentially) infinite rewrite relations, for two natural generalisation of acyclicity, termination respectively cotermination.

4.2 Preservation of termination by strong triangulation

To obtain completeness for infinite rewrite relations as well, we bar the counterexamples of the previous section by requiring, on top of the **(affluence)** and **(codeterminism)**, the following strengthening of **(triangulation)**

$$\mathrel{-\!\!\blacktriangleright}\ \subseteq ((\leftarrowtriangle \cdot \to) \cup (\leftarrow \cdot \rightarrowtriangle)) \cap (\leftarrowtriangle\!\!\!\triangleleft \cdot ((\leftarrowtriangle \cdot \rightarrowtriangle) - \mathsf{id}) \cdot \rightarrowtriangle\!\!\!\triangleright) \qquad \textbf{(strong triangulation)}$$

The first conjunct of **(strong triangulation)** captures the idea that every $\mathrel{-\!\!\blacktriangleright}$-step is caused by a peak containing at least one $\rightarrowtriangle$-step, while the second conjunct captures that the $\mathrel{-\!\!\blacktriangleright}$-step originates with some non-trivial peak of $\rightarrowtriangle$-reductions. We proved that **(strong triangulation)** captures an essential aspect of triangulation in the sense that for any *codeterministic* rewrite relation its triangulation satisfies **(strong triangulation)**; the proof is found in the report version [7] of this paper.

▶ Remark. That **(strong triangulation)** is a proper strengthening of **(triangulation)** can be seen by considering the rewrite relations in Counterexamples 4.4. That the triangulation of a *non-codeterministic* rewrite relation may fail to satisfy **(strong triangulation)** can be seen by viewing $\mathrel{-\!\!\blacktriangleright}$ in Example 4.1 as arising from triangulating $\rightarrowtriangle$, and considering the step $a \mathrel{-\!\!\blacktriangleright} e$.

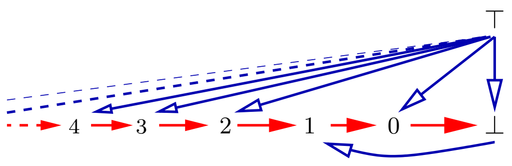


Figure 11 Non-terminating example satisfying all conditions of Corollary 4.6 except for the *first* conjunct of the **(strong triangulation)** property.

▶ **Theorem 4.5.** *For rewrite relations* $\mathrel{-\!\!\triangleright}$, $\mathrel{-\!\!\blacktriangleright}$ *and* $\to\; = \;\mathrel{-\!\!\triangleright} \cup \mathrel{-\!\!\blacktriangleright}$, *if properties* ***(strong triangulation)*** *and* ***(codeterminism)*** *hold and* $\mathrel{-\!\!\triangleright}$, $\mathrel{-\!\!\blacktriangleright}$ *are terminating, then* $\to$ *is terminating.*

Proof. We proceed as in the proof of Theorem 3.6. That is, to prove termination of $\to$ under the assumption that $\mathrel{-\!\!\triangleright}$, $\mathrel{-\!\!\blacktriangleright}$ are terminating, it suffices by Lemma 3.5 to prove $\mathrel{-\!\!\triangleright}^+ \cdot \mathrel{-\!\!\blacktriangleright} \;\subseteq\; \mathrel{-\!\!\triangleright}^+ \cup (\mathrel{-\!\!\blacktriangleright} \cdot \twoheadrightarrow)$. The latter property follows from the following claim.

Claim: If $a \mathrel{-\!\!\triangleright}^+ b \mathrel{\blacktriangleleft\!\!-\!\!\blacktriangleright} c$ then $a\ (\mathrel{-\!\!\triangleright}^+ \cup (\mathrel{-\!\!\blacktriangleright} \cdot \twoheadrightarrow))\ c$, for all a, b, c.

We prove this claim by induction on $n+m$ for the natural numbers n, m such that it holds $b\ (\mathrel{\triangleleft\!\!-}^n \cdot ((\mathrel{\triangleleft\!\!-} \cdot \mathrel{-\!\!\triangleright}) - \mathsf{id}) \cdot \mathrel{-\!\!\triangleright}^m)\ c$. Note that the pair exists by the second conjunct of the **(strong triangulation)** property, and that it is unique by the **(codeterminism)** property and the assumption that $\mathrel{-\!\!\triangleright}$ is terminating. From the **(strong triangulation)** property for $b \mathrel{\blacktriangleleft\!\!-\!\!\blacktriangleright} c$ we conclude that there exists b' such that one of the following three cases holds.

- $b \mathrel{\triangleleft\!\!-} b' \mathrel{-\!\!\triangleright} c$. Then by the **(codeterminism)** property, $a \mathrel{-\!\!\triangleright\!\!\triangleright} b' \mathrel{-\!\!\triangleright} c$ and we conclude.
- $b \mathrel{\triangleleft\!\!-} b' \mathrel{-\!\!\blacktriangleright} c$. Then by the **(codeterminism)** property, $a \mathrel{-\!\!\triangleright\!\!\triangleright} b' \mathrel{-\!\!\blacktriangleright} c$. If $a = b'$ then we conclude. Otherwise we conclude by the induction hypothesis for $a \mathrel{-\!\!\triangleright}^+ b' \mathrel{-\!\!\blacktriangleright} c$, which applies since by the second conjunct of the **(strong triangulation)** property for $b' \mathrel{-\!\!\blacktriangleright} c$, for some pair n', m' of positive natural numbers $b'\ (\mathrel{\triangleleft\!\!-}^{n'} \cdot ((\mathrel{\triangleleft\!\!-} \cdot \mathrel{-\!\!\triangleright}) - \mathsf{id}) \cdot \mathrel{-\!\!\triangleright}^{m'})\ c$, hence also $b\ (\mathrel{\triangleleft\!\!-}^{n'+1} \cdot ((\mathrel{\triangleleft\!\!-} \cdot \mathrel{-\!\!\triangleright}) - \mathsf{id}) \cdot \mathrel{-\!\!\triangleright}^{m'})\ c$, so $n = n'+1$ and $m = m'$ by uniqueness of the pair n, m;
- $b \mathrel{\blacktriangleleft\!\!-} b' \mathrel{-\!\!\triangleright} c$. Then $a\ (\mathrel{-\!\!\triangleright}^+ \cup (\mathrel{-\!\!\blacktriangleright} \cdot \twoheadrightarrow))\ b'$ by the induction hypothesis for $a \mathrel{-\!\!\triangleright}^+ b \mathrel{\blacktriangleleft\!\!-} b'$, which applies for the same reason as in the previous item. From that we conclude again. ◀

▶ **Corollary 4.6.** *For rewrite relations* $\mathrel{-\!\!\triangleright}$, $\mathrel{-\!\!\blacktriangleright}$ *and* $\to\; = \;\mathrel{-\!\!\triangleright} \cup \mathrel{-\!\!\blacktriangleright}$, *if properties* ***(strong triangulation)*** *and* ***(codeterminism)*** *hold and* $\mathrel{-\!\!\triangleright}$, $\mathrel{-\!\!\blacktriangleright}$ *are terminating, then* $\to$ *is complete.* Neither conjunct of the **(strong triangulation)** property can be dispensed with.

▶ Counterexamples 4.7. The first example of Counterexamples 4.4 satisfies all conditions of Corollary 4.6 except for the *second* conjunct of the **(strong triangulation)** property, yet $\to$ is not complete; it is not terminating as we have seen before.

For an example satisfying all conditions of Corollary 4.6 except for the *first* conjunct of the **(strong triangulation)** property, consider the natural numbers extended with $\bot$,$\top$, where $\mathrel{-\!\!\triangleright}$ is given by $\bot \mathrel{-\!\!\triangleright} 1$, $\top \mathrel{-\!\!\triangleright} \bot$, and $\top \mathrel{-\!\!\triangleright} n$ for all $n \neq 1$, and where $\mathrel{-\!\!\blacktriangleright}$ is the greater-than relation on natural numbers extended with $0 \mathrel{-\!\!\blacktriangleright} \bot$ (see Figure 11 where transitive $\mathrel{-\!\!\blacktriangleright}$-edges have been omitted). That the conditions hold is easy to check, yet $\to$ is not complete; it is not terminating as it admits the cycle $1 \mathrel{-\!\!\blacktriangleright} 0 \mathrel{-\!\!\blacktriangleright} \bot \mathrel{-\!\!\triangleright} 1$. Although the example does not satisfy the first conjunct $\mathrel{-\!\!\blacktriangleright} \;\subseteq\; (\mathrel{\triangleleft\!\!-} \cdot \to) \cup (\leftarrow \cdot \mathrel{-\!\!\triangleright})$ of the **(strong triangulation)** property for $1 \mathrel{-\!\!\blacktriangleright} 0$, it does satisfy the weaker condition $\mathrel{-\!\!\blacktriangleright} \;\subseteq\; \leftarrow \cdot \to$.

4.3 Preservation of cotermination by triangulation

There are two natural generalisations of acyclicity from finite to infinite rewrite relations, termination and cotermination. In the previous section we have seen that for termination to be preserved **(triangulation)** had to be strengthened to **(strong triangulation)**.[7] In this section we show that for cotermination no such strengthening is needed: it is preserved without extra conditions. That is, investigating the properties of the triangulation process, we answer the dual question whether triangulating a *coterminating* rewrite relation $\mathrel{-\!\!\rhd}$ with respect to a *coterminating* relation R yields a *coterminating* rewrite relation, affirmatively.

That the triangulation of $\mathrel{-\!\!\rhd}$ with respect to R is coterminating if their union $\mathrel{-\!\!\rhd} \cup R$ is coterminating, follows by the same proof as the one of Theorem 3.2. The interesting situation is again when $\mathrel{-\!\!\rhd}$ is assumed to be codeterministic, but no compatibility or finiteness condition is put on the coterminating relations $\mathrel{-\!\!\rhd}, R$.[8] We first show an auxiliary result, characterising the $\mathrel{-\!\!\rhd}$-reduction peaks that cause $\to$-steps.

▶ **Lemma 4.8** (Δ). *For rewrite relations $\mathrel{-\!\!\rhd}$, $\mathrel{-\!\!\blacktriangleright}$ and $\to\, = \,\mathrel{-\!\!\rhd} \cup \mathrel{-\!\!\blacktriangleright}$, if properties* ***(triangulation)*** *and* ***(codeterminism)*** *hold and $\mathrel{-\!\!\rhd}$, $\mathrel{-\!\!\blacktriangleright}$ are coterminating, then for all $a_0 \to a_1$ a common $\mathrel{-\!\!\rhd}$-expansion $a^{0,1}$ of a_0,a_1 exists, such that $a_1 \mathrel{\blacktriangleleft\!\!-}^{+} b$, for $i \in \{0,1\}$ and all b with $a_i \mathrel{\lhd\!\!-}^{+} b \mathrel{\lhd\!\!-}^{+} a^{0,1}$ (see Figure 12).*

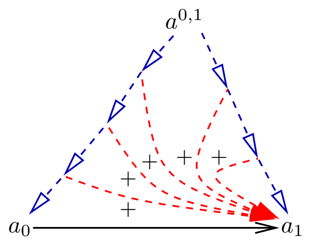


Figure 12 Illustration of the statement of the Δ-lemma.

Proof. The proof is by well-founded induction on the pair consisting of a_1 and the multiset of all $\mathrel{-\!\!\rhd}$-expansions of a_0, finite by **(codeterminism)** and cotermination of $\mathrel{-\!\!\rhd}$, well-foundedly ordered by the lexicographic product of $\mathrel{\blacktriangleleft\!\!-}$ and its multiset extension. We distinguish cases on the step $a_0 \to a_1$.

($a_0 \mathrel{-\!\!\blacktriangleright} a_1$) We distinguish cases on the peak obtained by **(triangulation)**:

- if $a_0 \mathrel{\lhd\!\!-} a_2 \mathrel{-\!\!\rhd} a_1$, then, setting $a^{0,1} = a_2$, we conclude immediately;
- if $a_0 \mathrel{\lhd\!\!-} a_2 \mathrel{-\!\!\blacktriangleright} a_1$, then we conclude by the induction hypothesis for $a_2 \mathrel{-\!\!\blacktriangleright} a_1$, which applies by a decrease in the second component as the multiset of $\mathrel{-\!\!\rhd}$-expansions of a_2 is a proper submultiset of that of a_0, since $a_0 \mathrel{\lhd\!\!-} a_2$;
- if $a_0 \mathrel{\blacktriangleleft\!\!-} a_2 \mathrel{-\!\!\rhd} a_1$, then we conclude by the induction hypothesis for $a_2 \mathrel{-\!\!\blacktriangleright} a_0$, which applies by a decrease in the first component, since $a_1 \mathrel{\blacktriangleleft\!\!-} a_0$;
- if $a_0 \mathrel{\blacktriangleleft\!\!-} a_2 \mathrel{-\!\!\blacktriangleright} a_1$ then the induction hypothesis applies first to $a_2 \mathrel{-\!\!\blacktriangleright} a_0$ yielding a common $\mathrel{-\!\!\rhd}$-expansion $a^{2,0}$ (decrease in the first component: $a_1 \mathrel{\blacktriangleleft\!\!-} a_0$) and next to

[7] Counterexamples 4.4 showed that the **(triangulation)** property was *not* sufficient to show preservation of *termination* in the case of infinite rewrite relations.

[8] By Corollary 3.8 cotermination is seen to be preserved in the case of *finite* rewrite relations, since then termination, acyclicity, and cotermination all coincide.

$a_2 \mathrel{-\!\!\blacktriangleright} a_1$ yielding a common $\rightarrowtriangle$-expansion $a^{2,1}$ (decrease in the second component: the objects on the $\rightarrowtriangle$-expansion from a_2 up to $a^{2,0}$ all are $\mathrel{-\!\!\blacktriangleright}$-expansions of a_0 via a_2). By **(codeterminism)** $a^{2,1-j} \mathrel{\lhd\!\!\lhd\!\!-} a^{2,j}$ for one of $j \in \{0,1\}$, so we may set $a^{0,1} = a^{2,j}$ as common $\rightarrowtriangle$-expansion of a_0,a_1. If $a_i \leftarrowtriangle^+ b \leftarrowtriangle^+ a^{0,1}$ for some $i \in \{0,1\}$ and some b, then either $a_i \leftarrowtriangle^+ b \leftarrowtriangle^+ a^{2,i}$ or $a_2 \mathrel{\lhd\!\!\lhd\!\!-} a^{2,i} \mathrel{\lhd\!\!\lhd\!\!-} b \leftarrowtriangle^+ a^{0,1} = a^{2,j}$ and we conclude by either induction hypothesis and $a_1 \mathrel{\blacktriangleleft\!\!-} a_0$;

($a_0 \rightarrowtriangle a_1$) Then we conclude, setting $a^{0,1} = a_0$, trivially. ◀

Nothing in the conditions of the Δ-lemma entails *completeness* of $\rightarrow$, and indeed it may fail to hold as can be seen by letting $\rightarrowtriangle$ be the (coterminating) predecessor relation on natural numbers and $\mathrel{-\!\!\blacktriangleright}$ the empty relation. We do however have *cocompleteness* (coconfluence and cotermination) even without requiring **(affluence)**.

▶ **Theorem 4.9.** *For rewrite relations $\rightarrowtriangle$, $\mathrel{-\!\!\blacktriangleright}$ and $\rightarrow = \rightarrowtriangle \cup \mathrel{-\!\!\blacktriangleright}$, if properties **(codeterminism)** and **(triangulation)** hold and $\rightarrowtriangle$, $\mathrel{-\!\!\blacktriangleright}$ are coterminating, then $\rightarrow$ is cocomplete.*

Proof. First we consider cotermination of $\rightarrow$. For a proof by contradiction, suppose $a_1 \leftarrow a_2 \leftarrow \dots$ were an infinite expansion. Then adjoining an object a_0, and steps $a_{i+1} \mathrel{-\!\!\blacktriangleright} a_0$, for all i, would yield relations still satisfying the assumptions. The Δ-lemma yields a contradiction as a_0 and a_1 do not have a common $\rightarrowtriangle$-expansion.[9]

For the proof of coconfluence of $\rightarrow$, note that the Δ-lemma yields that $\rightarrow$-convertibility is contained in $\rightarrowtriangle$-convertibility, from which one concludes as $\rightarrowtriangle$ is trivially coconfluent by **(codeterminism)**. ◀

Although we think the proof of the result, in particular that of the Δ-lemma, is interesting, as of yet this is just a theoretical result and we do not have applications (except for providing an alternative more general/complex proof to Corollary 4.3). Still, the result lends itself well for a reformulation as a nice puzzle, i.e. one that is easy to understand but hard to solve.

▶ Puzzle 4.10. Consider a city with Red ($\mathrel{-\!\!\blacktriangleright}$) and Blue ($\rightarrowtriangle$) bus lines (see Figure 13 left):

- Blue buses are *deterministic*, i.e. the next stop of a Blue bus (if it can go anywhere at all) is completely determined by the stop it's currently at;
- Red buses can be *triangulated*, i.e. if a Red bus can go directly from stop a to stop b, then there is a stop c (not necessarily distinct from a,b) such that one can go directly from both a and b to c, in each case by either taking a Red or a Blue bus.

Show that if one can make an infinite trip using buses of either company, then one can make an infinite monochrome trip, i.e. a trip using buses of one and the same company only.[10]

To solve Puzzle 4.10 for the particular case displayed in Figure 13 it suffices to apply, the contrapositive of, the construction in the proof of the Δ-lemma (for the converse of the relations):

▶ Solution 4.11. Applying the construction to the bus route from top–right to top–left in the circuit on the left in Figure 13 leads to adjoining an infinite succession of triangles to $\mathrel{-\!\!\blacktriangleright}$-steps, as indicated by the numbers on the right in the figure, and thereby to an infinite monochrome trip, the Red trip indicated by the thick arrows.

A general solution to the puzzle requires considering arbitrary length bichrome trips instead of single hops, corresponding exactly to our result.

[9] For an alternative constructive proof, one may show, measuring objects by their multiset of $\rightarrowtriangle$-expansions, a decrease in the multiset extension of $\mathrel{\blacktriangleleft\!\!-}$, along any step.

[10] So one can save lots of money by buying an ∞-*pass* from one of the companies only.

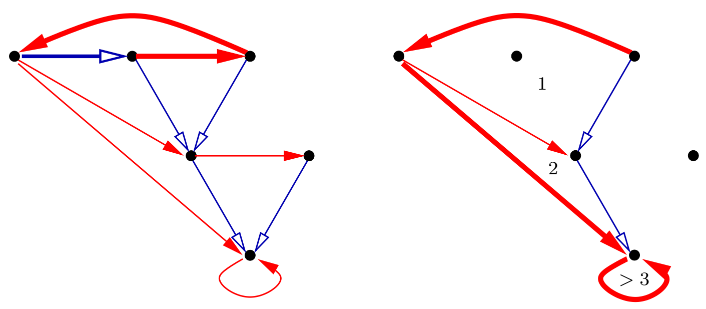


Figure 13 On the left Red and Blue bus lines satisfying the conditions of the puzzle are shown. On the right it is shown how to construct an infinite monochrome Red trip (the thick arrows).

5 Conclusion

In diagram completion in rewriting theory [6] the focus is on square diagrams. For instance, for proving confluence if a rewrites to both b and c we are looking for d such that both b and c rewrite to d, as depicted in a square diagram as is in the RTA logo. In contrast, in this paper not these squares are the basic building blocks, but the even more basic triangles: in the above setting we directly want to relate b and c. Inspired by computational geometry where splitting up polygons into triangles is called triangulation, we defined triangulation to be the process of extending a rewrite relation by extending every peak to a triangle. In this paper we studied some basic properties of triangulation, and its relation to confluence and termination. It turned out that the result of triangulation is always confluent (Theorem 2.5), even affluent, being the natural strengthening of confluence with respect to triangulation. We showed that in general triangulation does not preserve termination (Example 3.1), but that in case the initial rewrite relation is codeterministic termination is preserved (Theorem 3.6); the proof of this property was remarkably hard.

Acknowledgments. We thank Ashish Tiwari, Nachum Dershowitz, the referees, and the attendants of TeReSe and the TF-lunch seminar, for feedback and interesting discussions.

References

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

2 L. Bachmair and N. Dershowitz. Equational Inference, Canonical Proofs, and Proof Orderings. *Journal of the ACM*, 41(2):236–276, 1994.

3 N. Dershowitz. On Lazy Commutation. In *Languages: From Formal to Natural*, volume 5533 of *Lecture Notes in Computer Science*, pages 59–82. Springer, 2009.

4 H. Doornbos and B. von Karger. On the Union of Well-Founded Relations. *Logic Journal of the IGPL*, 6(2):195–201, 1998.

5 J.F. Groote, J. Keiren, A. Mathijssen, B. Ploeger, F. Stappers, C. Tankink, Y. Usenko, M. van Weerdenburg, W. Wesselink, T. Willemse, and J. van der Wulp. The mCRL2 toolset. In *Proceedings of the International Workshop on Advanced Software Development Tools and Techniques (WASDeTT 2008)*, 2008.

6 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

7 H. Zantema and V. van Oostrom. Triangulation in rewriting. Logic Group Preprint Series 292, Utrecht University, 2012. `http://www.phil.uu.nl/preprints/lgps/`.

Turing-Completeness of Polymorphic Stream Equation Systems

Christian Sattler and Florent Balestrieri

School of Computer Science, University of Nottingham
Nottingham, NG8 1BB, UK
`{cvs,fyb}@cs.nott.ac.uk`

Abstract

Polymorphic stream functions operate on the structure of streams, infinite sequences of elements, without inspection of the contained data, having to work on all streams over all signatures uniformly. A natural, yet restrictive class of polymorphic stream functions comprises those definable by a system of equations using only stream constructors and destructors and recursive calls. Using methods reminiscent of prior results in the field, we first show this class consists of exactly the computable polymorphic stream functions. Using much more intricate techniques, our main result states this holds true even for unary equations free of mutual recursion, yielding an elegant model of Turing-completeness in a severely restricted environment and allowing us to recover previous complexity results in a much more restricted setting.

1998 ACM Subject Classification F.1.1 Models of Computation; F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases stream functions, Turing-completeness, recursion

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.256

Category Regular Research Paper

1 Introduction

Streams over some set $\mathbb{D}$ are the basic example of a polymorphic coinductive data type, having been forced to serve as case study for almost every technique dealing with infinite data structures. Although being well-explored coalgebraic objects [5, 3, 19], they have recently re-emerged in the specific setting of term rewriting [22, 6]. They are usually introduced as the coinductive data type $\mathrm{Str}_{\mathbb{D}}$ generated by the constructor $\cdot :: \cdot : \mathbb{D} \times \mathrm{Str}_{\mathbb{D}} \to \mathrm{Str}_{\mathbb{D}}$ (cons), appending an element to the front of a stream, and come with destructors $\mathrm{head} : \mathrm{Str}_{\mathbb{D}} \to \mathbb{D}$ and $\mathrm{tail} : \mathrm{Str}_{\mathbb{D}} \to \mathrm{Str}_{\mathbb{D}}$, selecting and removing the front element, respectively. Algebraically, they can be characterised as an infinite term model parameterised by the value type $\mathbb{D}$ modulo observational equivalence on $\mathbb{D}$.

Since this work is concerned with computability and partiality, we choose to work in a semantics of partial streams, adding a bottom element $\perp$ to the underlying data type. Technically, such streams are just functions of type $\mathbb{N} \to \mathbb{D}_\perp$. Note that with this terminology, if an element of a stream equals $\perp$, further elements can still be proper inhabitants of $\mathbb{D}$. Also, when speaking of computable (stream) functions, we always mean partial computable (stream) functions.

One of the simplest classes of functions on streams are the polymorphic stream functions, being parametric in the data type $\mathbb{D}$. This prohibits any kind of pattern matching or case distinction on the underlying data type, effectively restricting them to discarding, duplicating, and reordering of the input stream elements. This defines an indexing function which in the unary case has type $\mathbb{N} \to \mathbb{N}_\perp$, associating with each output stream index the corresponding input stream index to copy from, or $\perp$ if the output element is $\perp$. We can consider a polymorphic stream function f computable if this indexing function, denoted $\overline{f}$, is computable. What we call indexing function is a container morphism for streams in the terminology of Abott et al. [1].

We consider recursive stream equation systems for specifying polymorphic stream functions involving only stream constructors and destructors. As a representative example, consider the system

$$\begin{aligned} \mathrm{const}(s) &= \mathrm{head}(s) :: \mathrm{const}(s), \\ \mathrm{zip}_2(s, t) &= \mathrm{head}(s) :: \mathrm{zip}_2(t, \mathrm{tail}(s)), \\ \mathrm{hanoi}(s) &= \mathrm{zip}_2(\mathrm{hanoi}(\mathrm{tail}(s)), \mathrm{const}(s)). \end{aligned}$$

Through evaluation, which will be elaborated upon in the examples subsection, we find that

$$\mathrm{hanoi}(s) = \perp :: s(0) :: s(1) :: s(0) :: s(2) :: s(0) :: s(1) :: s(0) :: s(3) :: \ldots.$$

The corresponding indexing function is $\overline{\mathrm{hanoi}}(k) = \max\{v \text{ such that } 2^v \text{ divides } k\}$ where $\max\{\mathbb{N}\} := \perp$. To explain the naming, if $\mathbb{D}$ is instantiated with the set of disks of an infinite Tower of Hanoi and $s \in \mathrm{Str}_\mathbb{D}$ is a list of the disks sorted by increasing size, then $\mathrm{tail}(\mathrm{hanoi}(s))$ is a walkthrough for coinductively solving the puzzle, the k-th stream element being the disk to be moved in the k-th step, with the smallest disk always moving in the same direction [12].

The key point to stress is that polymorphism is a severe restriction. Constructing examples less trivial than the above seems out of reach: the reader is invited to try to encode the Fibonacci sequence as the indexing function of an equation in a polymorphic system.

Most proofs of undecidability and complexity results for stream equations, like the ones of Roşu [18] and Simonsen [20], use straightforward encodings of Turing machines, representing the infinite band of symbols as two streams, one each for the left and right side of the band relative to the head, with canonical rewrite rules pattern matching on the current symbol. This central dispatching mechanism is unavailable in our setting. Still, we are able to recover all of these results even in the unary setting as direct corollaries of Theorem 14.

As a first taste, Proposition 3 states that our limited systems are nevertheless still sufficient to define every computable polymorphic stream function. Although the construction requires some imagination, the simulation of counter machines is quite direct and mainly intended to give the reader some intuition for the long road towards the proof of our key result, Theorem 14, which improves upon this by restricting systems to unary stream functions without mutual recursion. This is the main contribution of our work, which seems surprising giving the crippled expressiveness of the syntax.

Endrullis et al. [7, 8, 9] strive to decompose rewriting into a stream layer and a data layer in such a way as to encapsulate just so much complexity into the data layer that the productivity of streams becomes decidable while still retaining usefulness of computation. Our work can be seen as a another extreme, eradicating the data layer and showing that polymorphic unary stream functions attain computational completeness. For example, our results imply that the lazy stream formats of Endrullis et al. [8] can actually be restricted to (general) unary stream functions with productivity still retaining Π^0_2-completeness (in the non-unary case, a hint of Proposition 3 can be found in their encoding of FRACTRAN-programs). We note that their notions of lazy stream specifications and data-oblivious analysis shares some points with our polymorphism restriction: choosing the unit type for the data type leaves no possibility of analysing the input. We also note that the flat stream specifications, for which the authors develop an algorithm for semi-deciding productivity, present an exception: we allow general nested calls.

See Simonsen [20] for a good survey of some of the complexity analysis on stream rewriting our developments generalise. We hope that our Turing-complete unary recursive systems, in their simplicity, may be used as a computational model in further reduction proofs (e.g. of complexity results) not only in rewriting theory. We conclude by remarking that all our proofs are constructive, i.e. algorithmically implementable.

2 Syntax and Semantics

2.1 Streams and Indexing Functions

Given a set $\mathbb{D}$, we denote $\mathbb{D}_\perp := \mathbb{D} \cup \{\perp\}$ the set together with a distinguished *bottom element* $\perp$, which is used to denotate partial or non-terminating computation on the object-level of stream reduction. We warn that on the meta-level, $\perp$ is in quoted form, i.e. treated as a normal element. Given a function $f : A \to B_\perp$, we adopt the convention of implicitly extending the domain of f to $A_\perp$ by setting $f(\perp) = \perp$.

The set of (partial) streams over $\mathbb{D}$ is defined as $\mathrm{Str}_\mathbb{D} := \mathbb{N} \to \mathbb{D}_\perp$. A stream s is *total* if $s(k) \neq \perp$ for all $k \in \mathbb{N}$. A *polymorphic stream function* is a family of functions

$$f_\mathbb{D} : \mathrm{Str}_\mathbb{D} \times \ldots \times \mathrm{Str}_\mathbb{D} \to \mathrm{Str}_\mathbb{D}$$

natural (or *parametric*) in the domain argument $\mathbb{D}$, i.e. for all sets $\mathbb{D}_1, \mathbb{D}_2$ and functions $g : \mathbb{D}_1 \to (\mathbb{D}_2)_\perp$, we have

$$(\mathrm{map}\, g) \circ f_{\mathbb{D}_1} = f_{\mathbb{D}_2} \circ (\mathrm{map}\, g, \ldots, \mathrm{map}\, g)$$

where $\mathrm{map}\, g : \mathrm{Str}_{\mathbb{D}_1} \to \mathrm{Str}_{\mathbb{D}_2}, s \mapsto g \circ s$. Such a function is called *total* if total arguments yield total results.

Even though a polymorphic stream function is an operation on streams, its parametricity property enables us to express it as a single stream on a particular domain. To see this, consider a stream function f of arity n. Let $I_n := \{0, \ldots, n-1\} \times \mathbb{N}$. In the following, we exploit the fact that the type of the argument of $f_\mathbb{D}$ is isomorphic to $I_n \to \mathbb{D}_\perp$. Define $\overline{f} : \mathbb{N} \to (I_n)_\perp$, $\overline{f} := f_{I_n}(j \mapsto (i,j))_{i \in \{0,\ldots,n-1\}}$. Essentially, this is f_{I_n} applied to the identity. Given an arbitrary domain $\mathbb{D}$ and stream arguments $s_0, \ldots, s_{n-1} \in \mathrm{Str}_\mathbb{D}$, define $g : I_n \to \mathbb{D}_\perp$, $(i,j) \mapsto s_i(j)$. By parametricity,

$$\begin{aligned} f_\mathbb{D}(s_0, \ldots, s_{n-1}) &= (f_\mathbb{D} \circ (\mathrm{map}\, g, \ldots, \mathrm{map}\, g))(j \mapsto (i,j))_{i \in \{0,\ldots,n-1\}} \\ &= ((\mathrm{map}\, g) \circ f_{I_n})(j \mapsto (i,j))_{i \in \{0,\ldots,n-1\}} \\ &= (\mathrm{map}\, g)(\overline{f}). \end{aligned}$$

This equation is the reason we call $\overline{f}$ the *indexing function* of f: for each $k \in \mathbb{N}$, the value of $f_\mathbb{D}(s_0, \ldots, s_{n-1})$ at stream position k is given by $s_i(j)$ if $\overline{f}(k) = (i,j)$, and $\perp$ if $\overline{f}(k) = \perp$, i.e.

$$f_\mathbb{D}(s_0, \ldots, s_{n-1}) = s_{\overline{f}(0)} :: s_{\overline{f}(1)} :: s_{\overline{f}(2)} :: \ldots$$

with the convention that $s_{(i,j)} := s_i(j)$ and $s_\perp = \perp$.

From this, it is clear that polymorphic stream functions and indexing functions of the same arity are in bijective correspondence. A polymorphic stream function is total if and only if its indexing function is total. We call it *computable* if its indexing function is a computable function. In the remainder of the article, we will identify I_1 with $\mathbb{N}$.

It is worth noting that indexing functions represent an inversion of the usual notions of input and output for stream functions. This contravariance is reflected in $\overline{f \circ g} = \overline{g} \circ \overline{f}$ for polymorphic $f_{\mathbb{D}}, g_{\mathbb{D}} : \mathrm{Str}_{\mathbb{D}} \to \mathrm{Str}_{\mathbb{D}}$, a fact heavily utilised in the rest of the article. Basic examples of polymorphic stream functions are the tail operation $\mathsf{tail}_{\mathbb{D}} : \mathrm{Str}_{\mathbb{D}} \to \mathrm{Str}_{\mathbb{D}}, s \mapsto i \mapsto s(i+1)$ with indexing function $\overline{\mathsf{tail}}(k) = k+1$ and the combined head-cons operation

$$(\mathrm{head}(\cdot) :: \cdot)_{\mathbb{D}} : \mathrm{Str}_{\mathbb{D}} \times \mathrm{Str}_{\mathbb{D}} \to \mathrm{Str}_{\mathbb{D}},$$
$$(s,t) \mapsto i \mapsto \begin{cases} s(0) & \text{if } i = 0, \\ t(i-1) & \text{else} \end{cases}$$

with indexing function

$$\overline{(\mathrm{head}(\cdot) :: \cdot)}(k) = \begin{cases} (0,0) & \text{if } k = 0, \\ (1, k-1) & \text{else.} \end{cases}$$

2.2 Stream Equation Systems

We now define a simple form of a recursive system of equations for specifying polymorphic stream functions. A *(stream equation) system* of size n is a family of *stream equations* $f_k(s_0, \ldots, s_{m_k-1}) = \sigma_k$ with $k \in \{0, \ldots, n-1\}$ where each σ_k is a *stream term* of the form

$$\begin{array}{lll} \sigma ::= & s_i & \text{stream parameter with } i \in \{0, \ldots, m_k - 1\}, \\ \quad\mid & \mathsf{tail}(\sigma) & \text{stream stripped of its first element,} \\ \quad\mid & \mathrm{head}(\sigma) :: \sigma' & \text{first element of a stream prepended to stream (head-cons),} \\ \quad\mid & f_j(\sigma_0, \ldots, \sigma_{m_j-1}) & \text{recursive stream function call with } j \in \{0, \ldots, n-1\}. \end{array}$$

We explicitly state that there are no further restrictions such as guardedness on the form of the stream terms σ_k since we specifically deal with ill-defined equation using our (domain theoretic) partiality semantics. A system is called *unary* if all defined stream functions are unary, i.e. $m_k = 1$ for all k. By a canonical application of the Kleene fixed-point theorem [10], each stream equation system of size n gives rise to n corresponding polymorphic stream functions, the least fixpoint of the given system of equations with respect to the partial ordering on $\mathbb{D}_\perp$ generated by $\perp < d$ for $d \in \mathbb{D}$ when the tail and head-cons operations are interpreted according to the previous section. An equation in the system is called *productive* if the polymorphic stream function it defines is total. In what follows, we will usually use the same symbols to denote syntactic occurrences and their semantic counterparts as their meaning is always clear from the context.

We can also view such a system as an *executable specification*, giving rise to a notion of operational semantics allowing us to explicitly construct the least fixpoint alluded to above. For this, we formalise the computation of stream elements using a functional relation $\to$ on pairs $\sigma \ ! \ k$ of stream terms σ and indices $k \in \mathbb{N}$ by setting

$$\mathsf{tail}(\sigma) \ ! \ k \to \sigma \ ! \ k+1,$$
$$\mathrm{head}(\sigma) :: \sigma' \ ! \ k \to \begin{cases} \sigma \ ! \ 0 & \text{if } k = 0, \\ \sigma' \ ! \ k-1 & \text{else,} \end{cases}$$
$$f_j(\sigma_0, \ldots, \sigma_{n_j-1}) \ ! \ k \to \rho[\sigma_i/s_i]_{i \in \{0, \ldots, n_j-1\}} \ ! \ k$$

where $f_j(s_0, \ldots, s_{n_j-1}) = \rho$ is an equation in the system. This is effectively equivalent to introducing a rewriting system on constructs of the form $\mathrm{head}(\mathsf{tail}^k(\sigma))$ based on the rules

$$\mathrm{head}(\mathrm{head}(\sigma) :: \sigma') \to \mathrm{head}(\sigma),$$
$$\mathsf{tail}(\mathrm{head}(\sigma) :: \sigma') \to \sigma'$$

with a deterministic outermost rewriting strategy. For a given domain $\mathbb{D}$, we can now define $f_{j,\mathbb{D}}$ as

$$f_{j,\mathbb{D}}(s_0,\dots,s_{m_j-1}) = \begin{cases} s_w(i) & \text{if } f_j(s_0,\dots,s_{m_j-1}) \ ! \ k \to^* s_w \ ! \ i, \\ \bot & \text{else} \end{cases}$$

for $j \in \{0,\dots,n-1\}$, verifying that this is indeed the smallest solution to the given specification and fulfills the parametricity property. From this, we can express the indexing function as

$$\overline{f_j}(k) = \begin{cases} (w,i) & \text{if } f_j(s_0,\dots,s_{m_j-1}) \ ! \ k \to^* s_w \ ! \ i, \\ \bot & \text{else.} \end{cases}$$

Retrospectively, this consolidates our definition of productivity with its usual connotation, namely that each finite prefix of a stream (or the result of a stream function called with productive arguments) be constructible through finite evaluation. It furthermore shows that in this restricted settings, what is usually denoted by productivity is equivalent to unique solvability for instantiations of the parameter $\mathbb{D}$ to sets containing more than one element, i.e. well-definedness. On a side note, this description of indexing functions makes explicit the obvious fact that the defined stream functions are always computable.

2.3 Examples

Using this syntax, we can specify the *interleaving* function zip_n for $n > 0$ as

$$\text{zip}_n(s_0,\dots,s_{n-1}) = \text{head}(s_0) :: \text{zip}_n(s_1,\dots,s_{n-1},\text{tail}(s_0)).$$

It is productive with indexing function $\overline{\text{zip}_n}(k) = (k \bmod n, \lfloor k/n \rfloor)$. Let us prove this statement in detail by induction. In the base case, we have

$$\text{zip}_n(s_0,\dots,s_{n-1}) \ ! \ 0 \to \text{head}(s_0) :: \text{zip}_n(s_1,\dots,s_{n-1},\text{tail}(s_0)) \ ! \ 0 \to s_0 \ ! \ 0$$

proving $\overline{\text{zip}_n}(0) = (0,0) = (0 \bmod n, \lfloor 0/n \rfloor)$. In the induction step, we have

$$\begin{aligned} \text{zip}_n(s_0,\dots,s_{n-1}) \ ! \ k+1 &\to \text{head}(s_0) :: \text{zip}_n(s_1,\dots,s_{n-1},\text{tail}(s_0)) \ ! \ k+1 \\ &\to \text{zip}_n(s_1,\dots,s_{n-1},\text{tail}(s_0)) \ ! \ k \\ &\to^* \begin{cases} \text{tail}(s_0) \ ! \ \lfloor k/n \rfloor & \text{if } k \equiv -1 \mod n, \\ s_{(k \bmod n)+1} \ ! \ \lfloor k/n \rfloor & \text{else} \end{cases} \\ &\to^* s_{(k+1) \bmod n} \ ! \ \lfloor (k+1)/n \rfloor, \end{aligned}$$

proving $\overline{\text{zip}_n}(k) = (k \bmod n, \lfloor k/n \rfloor)$ implies $\overline{\text{zip}_n}(k) = ((k+1) \bmod n, \lfloor (k+1)/n \rfloor)$.

As a kind of inverse to interleaving, the *projection* function proj_n is defined as

$$\text{proj}_n(s) = \text{head}(s) :: \text{proj}_n(\text{tail}^n(s)).$$

It is easily shown to be productive with indexing function $\overline{\text{proj}_n}(k) = n{\cdot}k$. For convenience, we also define shifted projections $\text{proj}_{n,i}(s) := \text{proj}_n(\text{tail}^i(s))$ for $i < n$ with $\overline{\text{proj}_{n,i}}(k) = n \cdot k + i$. Note that $\text{proj}_{n,i,\mathbb{D}}(\text{zip}_{n,\mathbb{D}}(s_0,\dots,s_{n-1})) = s_i$ as well as $s = \text{zip}_{n,\mathbb{D}}\left((\text{proj}_{n,i,\mathbb{D}}(s))_{i\in\{0,\dots,n-1\}}\right)$ for all domains $\mathbb{D}$ and $s, s_0,\dots,s_{n-1} \in \text{Str}_{\mathbb{D}}$. We also have the *constant function* $\text{const}(s) = \text{head}(s) :: \text{const}(s)$, repeating the first stream element of its argument, with $\overline{\text{const}}(k) = 0$.

We are now ready to return to the example from the introduction. Recall that $\text{hanoi}(s) = \text{zip}_2(\text{hanoi}(\text{tail}(s)), \text{const}(s))$. We will prove that $\overline{\text{hanoi}}(2^v(2m+1)) = v$ by induction on v. In the base case, we have

$$\text{hanoi}(s) \ ! \ 2k+1 \to \text{zip}_2(\text{hanoi}(\text{tail}(s)), \text{const}(s)) \ ! \ 2k+1 \to^* \text{const}(s) \ ! \ k \to^* s \ ! \ 0,$$

proving $\overline{\text{hanoi}}(2k+1) = 0$. In the induction step, we have

$$\begin{aligned}\text{hanoi}(s) \ ! \ 2^{v+1}(2m+1) &\to \text{zip}_2(\text{hanoi}(\text{tail}(s)), \text{const}(s)) \ ! \ 2^{v+1}(2m+1) \\ &\to^* \text{hanoi}(\text{tail}(s)) \ ! \ 2^v(2m+1) \\ &\to^* \text{tail}(s) \ ! \ v \to s \ ! \ v+1,\end{aligned}$$

proving $\overline{\text{hanoi}}(2^v(2m+1)) = v$ implies $\overline{\text{hanoi}}(2^{v+1}(2m+1)) = v+1$. Now, the interesting artifact is the first stream position,

$$\text{hanoi}(s) \ ! \ 0 \to \text{zip}_2(\text{hanoi}(\text{tail}(s)), \text{const}(s)) \ ! \ 0 \to^+ \text{hanoi}(\text{tail}(s)) \ ! \ 0,$$

leading to an infinite loop

$$\text{hanoi}(s) \ ! \ 0 \to^+ \text{hanoi}(\text{tail}(s)) \ ! \ 0 \to^+ \text{hanoi}(\text{tail}^2(s)) \ ! \ 0 \to^+ \ldots,$$

showing that $\overline{\text{hanoi}}(0) = \bot$.

After having established some intuition for computing indexing functions, let us prove a lemma which will be of much use further on.

▶ **Lemma 1.** *Given $h \geq 1$ and a stream equation of the form*

$$f(s) = \text{head}(\text{tail}^{a_0}(s)) :: \ldots :: \text{head}(\text{tail}^{a_{h-1}}(s)) :: f(v(s)),$$

then $\overline{f}(k) = \overline{v}^{\lfloor k/h \rfloor}(a_{k \bmod h})$ for $k \in \mathbb{N}$.

Proof. Since this is our first technical result about indexing functions, we will be explicit in every detail. The proof is by induction on $k \in \mathbb{N}$. For $k < h$, we have

$$\begin{aligned}f(s) \ ! \ k &= \text{head}(\text{tail}^{a_0}(s)) :: \ldots :: \text{head}(\text{tail}^{a_{h-1}}(s)) :: f(v(s)) \ ! \ k \\ &\to^k \text{tail}^{a_k}(s) \ ! \ 0 \to^{a_k} s \ ! \ a_k,\end{aligned}$$

yielding $\overline{f}(k) = a_k = \overline{v}^0(a_k) = \overline{v}^{\lfloor k/h \rfloor}(a_{k \bmod h})$. For $k \geq h$, we have

$$f(s) \ ! \ k = \text{head}(\text{tail}^{a_0}(s)) :: \ldots :: \text{head}(\text{tail}^{a_{h-1}}(s)) :: f(v(s)) \ ! \ k \to^h f(v(s)) \ ! \ k-h.$$

By induction hypothesis, it follows that

$$\begin{aligned}\overline{f}(k) &= (\overline{f \circ v})(k-h) = \overline{v}(\overline{f}(k-h)) \\ &= \overline{v}(\overline{v}^{\lfloor (k-h)/h \rfloor}(a_{(k-h) \bmod h})) \\ &= \overline{v}(\overline{v}^{\lfloor k/h \rfloor - 1}(a_{k \bmod h})) = \overline{v}^{\lfloor k/h \rfloor}(a_{k \bmod h}),\end{aligned}$$

where we exploited contravariance of the indexing operation in the second step. ◀

3 Definability

Our first result consists of the insight that the above stream equations of simple form, incorporating only the stream constructor and destructors and recursion, already allow for the definition of every computable polymorphic stream function. In the following, we will only consider single argument functions since we can easily fuse multiple arguments into a single one using instances of zip and proj. The argument can be seen as a generalization of the idea expressed in the proof of Theorem 4.4 in an article by Endrullis et. al. [8] that recognising productivity of a form of system similar to ours (with a unit stream data type instead of abstract polymorphism) is of complexity Π^0_2. Since our key contribution concerns the even more general result of unary definability, the main purpose of the following exposition is to serve as a contrast to the next section.

Recall that for a single argument stream function, the indexing function has type $\mathbb{N} \to \mathbb{N}_\perp$. We will give our proof in the form of a reduction, transforming a counter machine [16] representing an arbitrary computable function $\phi : \mathbb{N} \to \mathbb{N}_\perp$ into a corresponding system with an equation having ϕ as indexing function.

▶ **Definition 2.** A *counter machine* is given by a tuple (L, I) where L is the length of the program and I is a list $I_0, \ldots, I_{L-1}$ of instructions $\mathrm{inc}(r)$ with $r \in \mathbb{N}$, denoting increment of register r, and $\mathrm{jzdec}(r, l)$ with $r \in \mathbb{N}$ and $l \in \{0, \ldots, L\}$, denoting a jump to instruction l if register r is zero and a decrement of r otherwise.

The semantics of such a machine is as follows: The *state* $(P, R) \in S := \{0, \ldots, L\} \times \mathbb{N}^{(\mathbb{N})}$ consists of the value P of its instruction pointer and the values R of its registers, where $\mathbb{N}^{(\mathbb{N})}$ denotes the set of functions $\mathbb{N} \to \mathbb{N}$ with finite support, i.e. with only finitely many values non-zero. Such a tuple is called *terminal* if $P = L$, denoting the machine has exited with output $R(1)$. For $R \in \mathbb{N}^{(\mathbb{N})}$ and $r, v \in \mathbb{N}$, the result of replacing the r-th entry of R with v will be denoted $R[r \leftarrow v]$. Representing execution, We define a functional *next* relation $\to$ on S on non-terminal elements by

$$(P,R) \to \begin{cases} (P+1, R[r \leftarrow R(r)+1]) & \text{if } I_P = \mathrm{inc}(r), \\ (P+1, R[r \leftarrow R(r)-1]) & \text{if } I_P = \mathrm{jzdec}(r,l), R(r) \neq 0, \\ (l, R) & \text{if } I_P = \mathrm{jzdec}(r,l), R(r) = 0. \end{cases}$$

The *result function* $\mathrm{result}_{(L,I)} : S \to \mathbb{N}_\perp$ is defined as

$$\mathrm{result}_{(L,I)}(t) = \begin{cases} R(1) & \text{for } t \to^* (L,R) \text{ terminal}, \\ \perp & \text{else.} \end{cases}$$

The *initial state* for a given input $i \in \mathbb{N}$ is given by $\mathrm{init}(i) := (0, (i, 0, \ldots))$. With this machinery, we can now define the associated computable function of the counter machine as $\phi_{(L,I)} = \mathrm{result}_{(L,I)} \circ \mathrm{init} : \mathbb{N} \to \mathbb{N}_\perp$.

▶ **Proposition 3.** *Given a computable function represented as a counter machine (L, I), there is a stream equation system defining a unary stream function with indexing function $\phi_{(L,I)}$.*

▶ **Definition 4.** Let $p_0 < p_1 < \ldots$ be all the primes. We encode a register state $R \in \mathbb{N}^{(\mathbb{N})}$ as a single positive integer using $\widehat{R} := \prod_{r \in \mathbb{N}, R(r) \neq 0} p_r^{R(r)}$.

Proof of Proposition 3. Our goal is to mutually define unary stream functions $f_0, \ldots, f_{L-1}$ such that for a sequence of machine states $(P, R) \to (P', R')$, we have $f_P \;!\; \widehat{R} \to^+ f_{P'} \;!\; \widehat{R'}$, effectively simulating the execution of the counter machine. It follows that whenever the counter machine terminates with $(P, R) \to^* (L, R')$ terminal, then $\overline{f_P} \;!\; \widehat{R} \to^* \overline{f_L} \;!\; \widehat{R'}$, and $\overline{f_P}(\widehat{R}) = \perp$ otherwise. Defining f_L to extract the value of register 1, i.e. the exponent of $p_1 = 3$, from the encoding we set

$$f_L(s) = \mathrm{zip}_3(f_L(\mathrm{tail}(s)), \mathrm{const}(s), \mathrm{const}(s)).$$

A straightforward induction on $R(1)$ shows that $\overline{f_L}(\widehat{R}) = R(1) = \mathrm{result}_{(L,I)}(L, R) \neq \perp$ for $R \in \mathbb{N}^{(\mathbb{N})}$. Together, this means $f_P(\widehat{R}) = \mathrm{result}_{(L,I)}(P, R)$ for any state $(P, R) \in S$.

For each instruction I_P with $P \in \{0, \dots, L-1\}$, we mutually define a corresponding stream function f_P reproducing the action of I_P on the register encoding. If $I_P = \mathrm{inc}(r)$, let

$$f_P(s) = \mathrm{proj}_{p_r}(f_{P+1}(s))$$

and note that $f_P \,!\, \widehat{R} \to^+ f_{P+1} \,!\, p_r\widehat{R} = f_{P+1} \,!\, R[r \leftarrow \widehat{R(r)+1}]$ for $R \in \mathbb{N}^{(\mathbb{N})}$, simulating an increment. If $I_P = \mathrm{jzdec}(r, l)$, let

$$f_P(s) = \mathrm{zip}_{p_r}\left(\left\{\begin{array}{ll} f_{P+1}(s) & \text{if i} = 0, \\ \mathrm{proj}_{p_r,i}(f_l(s)) & \text{else} \end{array}\right.\right)_{i\in\{0,\dots,p_r-1\}}.$$

Given $R \in \mathbb{N}^{(\mathbb{N})}$ with $R(r) \neq 0$, we know $R(r)$ is divisible by p_r and hence $f_P \,!\, \widehat{R} \to^+ f_{P+1} \,!\, \widehat{R}/p_r = f_{P+1} \,!\, R[r \leftarrow \widehat{R(r)-1}]$, simulating a decrement. For $R(r) = 0$, we have $f_P \,!\, \widehat{R} \to^+ f_l \,!\, p_r\lfloor\widehat{R}/p_r\rfloor + (\widehat{R} \bmod p_r) = f_l \,!\, \widehat{R}$, simulating a jump. Here, we exploited properties of the indexing functions of proj and zip noted earlier.

Finally, we need a stream function to produce the initial register encoding. This will be accomplished by $u(s) = \mathrm{head}(\mathrm{tail}(s)) :: u(\mathrm{proj}_{p_0}(s))$. By Lemma 1, we have $\overline{u}(i) = \overline{\mathrm{proj}_{p_0}}^{\,i}(1) = p_0^i = \widehat{(i, 0, \dots)}$. Defining $q(s) = u(f_0(s))$, we have $\overline{q}(i) = \overline{f_0}(\widehat{(i, 0, \dots)}) = \mathrm{result}_{(L,I)}(\mathrm{init}(i))$ for $i \in \mathbb{N}$, and thus $\overline{q} = \phi_{(L,I)}$. The equations for the stream functions $f_0, \dots, f_L, u, q,$ const plus finitely many instances of zip and proj hence represent a stream equation system with the denotation of q having $\phi_{(L,I)}$ as indexing function. ◀

Note that Minsky [16] shows that counter machines with only two registers are enough to achieve Turing-completeness. However, when representing computable functions, this requires an extra level of encoding of input values and decoding of output values, which can also be achieved by defining suitable stream functions.

4 Unary Definability

The defining feature of the previous construction was its reliance on interleaving for implementing conditional execution, effectively dispatching different cases, identified by their residues of the register encoding modulo a prime, to arbitrarily different handlers. Noting that zip_p with p prime were the only non-unary stream functions in the construction, we are left to reflect on the computational consequences of only allowing unary stream functions to be defined. Note that allowing interleaving is synonymous to allowing non-unary stream functions since we can use interleaving to merge any number of stream arguments into a single one. In order to prove an even more general definability result in the unary setting, entirely different techniques need to be developed, separating conditional execution and unbounded looping into orthogonal concepts.

4.1 Collatz Functions and If-Programs

▶ **Definition 5.** A function $g : \mathbb{N} \to \mathbb{N}$ is called a *Collatz function* if there is $n > 0$ such that g is affine on each equivalence class modulo n, i.e. there are coefficients $a_i, b_i \in \mathbb{N}$ for $i = 0, \dots, n-1$ such that $g(n \cdot q + i) = a_i \cdot q + b_i$ for $q \in \mathbb{N}$. In this case, n is called a *modulus* of g.

The naming stems from the famous conjecture first proposed by Collatz in 1937, asking whether the function collatz : $\mathbb{N} \to \mathbb{N}$, $n \mapsto n/2$ for n even, $n \mapsto 3n + 1$ for n odd, will map each positive integer to 1 after finitely many applications. Despite its deceivingly simple form, it has been resisting all attempts of resolution [15]. In recent years, the equivalent of this conjecture for the above generalised notion of Collatz functions has been proved (algorithmically) undecidable [14].

▶ **Lemma 6.** *Given a Collatz function g, we can construct a non-mutually recursive unary system defining a stream function v such that $\overline{v} = g$.*

Before going into the details of the proof, note that, although it is quite clear that the above encoding of Collatz functions already enables an embedding of full computational power into unary stream equations, what is not at all obvious is whether we can actually define every computable unary stream function through a purely unary system.

Proof. Let modulus $n > 0$ and coefficients $a_i, b_i \in \mathbb{N}$ be as in the above definition. Define a stream function

$$\mathrm{add}(s) = \mathrm{head}(\mathrm{tail}^{n \cdot a_0 + 0}(s)) :: \ldots :: \mathrm{head}(\mathrm{tail}^{n \cdot a_{n-1} + (n-1)}(s)) :: \mathrm{add}(\mathrm{tail}^n(s)).$$

Lemma 1 shows that $\overline{\mathrm{add}}(k) = \lfloor \frac{k}{n} \rfloor \cdot n + (n \cdot a_{k \bmod n} + (k \bmod n)) = k + n \cdot a_{k \bmod n}$ for $k \in \mathbb{N}$. The role of this function is to act as a crude replacement conditional for the unavailable zip, adding different constants depending on the equivalence class of the stream index modulo n.

Next, define a stream function

$$u(s) = \mathrm{head}(\mathrm{tail}^{n \cdot b_0 + 0}(s)) :: \ldots :: \mathrm{head}(\mathrm{tail}^{n \cdot b_{n-1} + (n-1)}(s)) :: u(\mathrm{add}(s)).$$

Using Lemma 1, we derive $\overline{u}(n \cdot q + i) = \overline{\mathrm{add}}^q(n \cdot b_i + i) = (n \cdot b_i + i) + q \cdot (n \cdot a_i) = n \cdot g(k) + i$ for $q \in \mathbb{N}$. This function is an approximation to g, the only difference being that the output indices come pre-multiplied by n.

We fix this by defining a stream function

$$\mathrm{div}(s) = \underbrace{\mathrm{head}(s) :: \ldots :: \mathrm{head}(s)}_{n \text{ times}} :: \mathrm{div}(\mathrm{tail}(s)).$$

A trivial application of Lemma 1 verifies that $\overline{\mathrm{div}}(k) = \lfloor \frac{k}{n} \rfloor$ for $k \in \mathbb{N}$. Finally defining $v(s) = u(\mathrm{div}(s))$ yields the Collatz function semantics we want in that $\overline{v} = \overline{\mathrm{div}} \circ \overline{u} = g$. ◀

For instance, the original Collatz function would be encoded as collatz $= \overline{v}$ as follows:

$$\begin{aligned}
\mathrm{add}(s) &= \mathrm{head}(\mathrm{tail}^2(s)) :: \mathrm{head}(\mathrm{tail}^{13}(s)) :: \mathrm{add}(\mathrm{tail}^2(s)),\\
u(s) &= \mathrm{head}(s) :: \mathrm{head}(\mathrm{tail}^9(s)) :: u(\mathrm{add}(s)),\\
\mathrm{div}(s) &= \mathrm{head}(s) :: \mathrm{head}(s) :: \mathrm{div}(\mathrm{tail}(s)),\\
v(s) &= u(\mathrm{div}(s)).
\end{aligned}$$

For further illustration, let us evaluate stream position 3 of $v(s)$:

$$\begin{aligned}
v(s) \;!\; 3 &\to^* u(\mathrm{div}(s)) \;!\; 3 \to^* u(\mathrm{add}(\mathrm{div}(s))) \;!\; 1 \to^* \mathrm{add}(\mathrm{div}(s)) \;!\; 9\\
&\to^* \mathrm{add}(\mathrm{tail}^2(\mathrm{div}(s))) \;!\; 7 \to^* \ldots \to^* \mathrm{add}(\mathrm{tail}^8(\mathrm{div}(s))) \;!\; 1\\
&\to^* \mathrm{div}(s) \;!\; 21 \to^* \mathrm{div}(\mathrm{tail}(s)) \;!\; 19 \to^* \ldots \to^* \mathrm{div}(\mathrm{tail}^{10}(s)) \;!\; 1\\
&\to^* s \;!\; 10.
\end{aligned}$$

This is consistent with collatz$(3) = 3 \cdot 3 + 1 = 10$.

Note that this encoding is fundamentally different from the encoding of Collatz functions or Fractran program iterations used by Endrullis et al. [8], the essential difference being the unavailability of zip in the unary setting. The dispatch mechanism of interleaving, enabling differing treatment of stream positions based on the residue of their indices, makes the implementation of Collatz-like constructs rather straightforward. Since we are lacking even such basic conditional control flow mechanisms, we have to resort to highly indirect constructions such as the above.

The role of Collatz functions in our setting is to serve as an intermediate between indexing functions and the known world of computability. To make the latter link clearer, we will show how Collatz functions relate semantically to different register machine models under the prime factorization register encoding introduced in the previous section.

▶ **Definition 7.** The inductive set of IF-*programs* is generated by concatenation $\mathbf{A}_0 \dots \mathbf{A}_{n-1}$, increments $\operatorname{inc}(r)$, decrements $\operatorname{dec}(r)$, and conditional clauses $\operatorname{ifz}(r, \mathbf{A}, \mathbf{B})$, where $n \in \mathbb{N}$, $r \in \mathbb{N}$ designates a register, and $\mathbf{A}_0, \dots, \mathbf{A}_{n-1}, \mathbf{A}, \mathbf{B}$ are IF-programs.

Although it is quite clear intuitively what the semantic effects of running an IF-program $\mathbf{A}$ on some register state $R \in \mathbb{N}^{(N)}$ are, we will formally introduce an associated semantics function $\chi_{\mathbf{A}} : \mathbb{N}^{(\mathbb{N})} \to \mathbb{N}^{(\mathbb{N})}$ defined structurally as follows:

$$\begin{aligned}
\chi_{\mathbf{A}_0 \dots \mathbf{A}_{n-1}} &= \chi_{\mathbf{A}_{n-1}} \circ \dots \circ \chi_{\mathbf{A}_0}, \\
\chi_{\operatorname{inc}(r)}(R) &= R[r \leftarrow R(r) + 1], \\
\chi_{\operatorname{dec}(r)}(R) &= R[r \leftarrow \max(R(r) - 1, 0)], \\
\chi_{\operatorname{ifz}(r,\mathbf{A},\mathbf{B})}(R) &= \begin{cases} \chi_{\mathbf{A}}(R) & \text{if } R(r) = 0, \\ \chi_{\mathbf{B}}(R) & \text{else.} \end{cases}
\end{aligned}$$

Note that a decrement on a zero-valued register is ignored.

We will reuse the prime factorization register encoding $\widehat{\cdot} : \mathbb{N}^{(\mathbb{N})} \to \mathbb{N} \setminus \{0\}, \widehat{R} \mapsto \widehat{R} = \prod_{r \in \mathbb{N}, R(r) \neq 0} p_r^{R(r)}$ from the previous section. Translated to this setting, the semantics function of an IF-program $\mathbf{A}$ takes the form $\widehat{\chi_{\mathbf{A}}} := \widehat{\cdot} \circ \chi_{\mathbf{A}} \circ \widehat{\cdot}^{-1}$. Although this is an endofunction on the positive integers, to make the following treatment more uniform, we will extend it to the natural numbers by setting $\widehat{\chi_{\mathbf{A}}}(0) := 0$.

▶ **Lemma 8.** *Given an if-program $\mathbf{A}$, its semantics $\widehat{\chi_{\mathbf{A}}} : \mathbb{N} \to \mathbb{N}$ on the register encoding is a Collatz function.*

Proof. By induction on the structure of $\mathbf{A}$, noting that:

- The concatenation of finitely many Collatz functions of moduli $m_0 \cdot \dots \cdot m_{n-1}$ is a Collatz function of modulus $m_0 \cdot \dots \cdot m_{n-1}$.
- Given a register state $R \in \mathbb{N}^{(\mathbb{N})}$, increment of register r corresponds to multiplication of $\widehat{R}$ with p_r, a Collatz function of modulus 1.
- Decrement of register r corresponds to division of $\widehat{R}$ by p_r if the former is divisible by p_r, and no change otherwise. This is a Collatz function of modulus p_r.
- Let $\widehat{\chi_{\mathbf{A}}}$ and $\widehat{\chi_{\mathbf{B}}}$ be Collatz functions of moduli $m_{\mathbf{A}}$ and $m_{\mathbf{B}}$, respectively. A conditional clause $\operatorname{ifz}(r, \mathbf{A}, \mathbf{B})$ corresponds first to case distinction depending on whether $\widehat{R}$ is divisible by p_r and subsequent application of either $\widehat{\chi_{\mathbf{A}}}$ or $\widehat{\chi_{\mathbf{B}}}$. This is a Collatz function of modulus the least common multiple of $p_r, m_{\mathbf{A}}, m_{\mathbf{B}}$.

◀

We note that the Collatz function $\widehat{\chi}_{\mathbf{A}}$ in the previous lemma is special in that it is linear on each of its equivalence classes in the strict sense, i.e. with vanishing ordinate, corresponding to single multiplication with a fraction. Even though we do not make use of this fact in our developments, it shows the connection between IF-programs and the iteration steps of the FRACTRAN-programs of Conway [4], which are of equivalent expressive power.

4.2 Iteration-Programs and Their Encoding

Unsurprisingly, the expressive power of IF-programs by themselves is quite limited. To achieve computational completeness, we need an unbounded looping construct. The following definition intends to provide a minimal such model, enabling us to concentrate on the essential details of the conversion from Turing-complete programs to stream equation systems.

▶ **Definition 9.** An ITERATION-*program* $\mathbf{P}$ is a tuple $(\mathbf{Body_P}, \mathsf{input_P}, \mathsf{output_P}, \mathsf{loop_P})$ consisting of an IF-program $\mathbf{Body_P}$ called the *body* of $\mathbf{P}$ and designated and mutually distinct *input*, *output* and *loop* registers $\mathsf{input_P}, \mathsf{output_P}, \mathsf{loop_P} \in \mathbb{N}$.

The semantics of such a program is a computable function $\phi_{\mathbf{P}} : \mathbb{N} \to \mathbb{N}_\bot$ defined as follows: given an input $i \in \mathbb{N}$, the register state $R_0 \in \mathbb{N}^{(\mathbb{N})}$ is initialised with $R_0(\mathsf{input_P}) := i$, $R_0(\mathsf{loop_P} := 1$, and $R_0(r) := 0$ for $r \neq \mathsf{input_P}, \mathsf{loop_P}$. We iteratively execute the body of $\mathbf{P}$, yielding $R_{n+1} := \chi_{\mathbf{Body_P}}(R_n)$ for $n \in \mathbb{N}$. If there is n minimal such that $R_n(\mathsf{loop_P}) = 0$, then $\mathbf{P}$ is called *terminating* with *iteration count* $\mathrm{count}\, P(i) := n$ and output $\phi_{\mathbf{P}}(i) := R_n(\mathsf{output_P})$ for input i. Otherwise, $\mathrm{count}_{\mathbf{P}}(i) := \bot$ and $\phi_{\mathbf{P}}(i) := \bot$.

Intuitively, an ITERATION-program is just a WHILE-program [17] with a single top-level loop, a well-studied concept in theoretical computer science bearing resemblance to the normal form theorem for μ-recursive functions [13], [21] except that we do not even allow primitive recursion inside the loop.

▶ **Theorem 10.** *Given a computable function $\phi : \mathbb{N} \to \mathbb{N}_\bot$, there is an* ITERATION-*program $\mathbf{P}$ with semantics $\phi_{\mathbf{P}} = \phi$.*

Proof. This is a folklore theorem [11], see Böhm and Jacopini [2] and Perkowska [17] for more details. ◀

The reason behind our choice for this computationally complete machine model is that we already have the machinery to simulate a single execution of the body of such a machine via Collatz functions as indexing functions of stream equations using our prime factorization exponential encoding on the register state. In fact, another option would have been the FRACTRAN-programs of Conway [4], but we are in need of a more conceptual representation in light of what lies ahead of us.

We will now investigate how to translate a top-level unbounded looping construct into the recursive stream equation setting.

▶ **Lemma 11.** *Let $h \geq 1$ be given with a stream equation*

$$f(s) = \mathrm{head}(\mathrm{tail}^{a_0}(s)) :: \ldots :: \mathrm{head}(\mathrm{tail}^{a_{h-1}}(s)) :: \mathrm{tail}^h(u(f(v(s)))).$$

Fix $k \in \mathbb{N}$ and choose $c(k) \in \mathbb{N}$ minimal such that $d(k) := \overline{u}^{c(k)}(k) \in \{\bot, 0, \ldots, h-1\}$. If such a $c(k)$ exists and $d(k) \neq \bot$, then $\overline{f}(k) = \overline{v}^{c(k)}(a_{d(k)})$, otherwise $\overline{f}(k) = \bot$.

Proof. The proof is by induction on $c(k)$ if existent. At the base, $c(k) = 0$ is equivalent to $k < h$. In this case, $f(s) \ ! \ k \to^k \mathrm{tail}^{a_k}(s) \ ! \ 0 \to^{a_k} s \ ! \ a_k$, i.e. $\overline{f}(k) = a_k = \overline{v}^{c(k)}(a_{d(k)})$.

Now assume $k \geq h$. Note that

$$f(s) \ ! \ k \to^h \operatorname{tail}^h(u(f(v(s)))) \ ! \ k - h \to^h u(f(v(s))) \ ! \ k.$$

If $\overline{u}(k) = \bot$, then $c(k) = 1$, $d(k) = \bot$, and $\overline{f}(k) = \bot$. In the remainder, we will assume $\overline{u}(k) \neq \bot$. Then, $f(s) \ ! \ k \to^+ f(v(s)) \ ! \ \overline{u}(k)$, and $\overline{f}(k) = \overline{v}(\overline{f}(\overline{u}(k)))$.

If $c(k)$ is defined, then $c(k) = c(\overline{u}(k)) + 1$ and we can apply the induction hypothesis: if $d(k) = d(\overline{u}(k)) \neq \bot$, then $\overline{v}(\overline{f}(\overline{u}(k))) = \overline{v}(\overline{v}^{c(\overline{u}(k))} a_{d(k)}) = \overline{v}^{c(k)}(a_{d(k)})$, otherwise $\overline{v}(\overline{f}(\overline{u}(k))) = \overline{v}(\bot) = \bot$.

If $c(k)$ is undefined, then so is $c(\overline{u}(k))$, and with a second induction we can construct an infinite sequence $f(s) \ ! \ k \to^+ f(v(s)) \ ! \ \overline{u}(k) \to^+ f(v^2(s)) \ ! \ \overline{u}^2(k) \to^+ \ldots$ showing non-termination and $\overline{f}(k) = \bot$. ◀

The inquiring reader will notice that this lemma can be seen as a generalization of Lemma 1 with u defined in a particular way, namely

$$u(s) = \underbrace{\operatorname{head}(s) :: \ldots :: \operatorname{head}(s)}_{h \text{ times}} :: s.$$

Using Lemmata 8 and 6, we can translate the encoded iteration step function $\widehat{\chi}_{\mathbf{Body_P}} : \mathbb{N} \to \mathbb{N}$ of an ITERATION-program $\mathbf{P}$ to an indexing function of a stream equation for some u. We would like to use this stream function u as it appears in Lemma 11 in a way such that the minimal choice of $c(k)$ corresponds to the iteration count of $\mathbf{P}$. Unfortunately, the equivalent of the stopping condition in the lemma, that the index be smaller than some constant h, corresponds to $\widehat{R} < h$ for the register state $R \in \mathbb{N}^{(\mathbb{N})}$, a statement which does not have a natural meaning for the registers of R individually, forestalling us from expressing the condition $p_{\mathsf{loop_P}} \mid \widehat{R}$ corresponding to the termination condition $R(\mathsf{loop_P}) = 0$. A second problem comes from our desire to somehow extract the value of $R(\mathsf{output_P})$ after termination. But since at this point of time $\widehat{R}$ is limited to a finite set of values, there is no direct way of realising this.

What we can do is extract the iteration count for particularly nicely behaving programs.

▶ **Lemma 12.** *Given an* ITERATION*-program* $\mathbf{Q}$ *such that whenever* $\mathbf{Q}$ *terminates, all its registers are zero-valued, i.e.* $\chi_{\mathbf{Body_Q}}^{\operatorname{count}_{\mathbf{Q}}(i)} = (0, 0, \ldots)$ *for terminating input* $i \in \mathbb{N}$*, there is a non-mutually recursive unary system defining a stream function* w *such that* $\overline{w} = \operatorname{count}_{\mathbf{Q}}$.

Proof. In anticipation of applying Lemma 11, we extend this system with a new equation

$$q(s) = \operatorname{head}(s) :: \operatorname{head}(s) :: \operatorname{tail}^2(v(q(\operatorname{tail}(s)))).$$

Given an input $i \in \mathbb{N}$ and corresponding initial register state $R \in \mathbb{N}^{(\mathbb{N})}$, the termination condition in Lemma 11 can equivalently be expressed as follows:

$$\overline{v}^{c(\widehat{R})}(\widehat{R}) < 2 \iff \widehat{\chi}_{\mathbf{Body_Q}}^{c(\widehat{R})}(\widehat{R}) = 1 \iff \chi_{\mathbf{Body_Q}}^{c(\widehat{R})}(R) = (0, 0, \ldots).$$

Now, by our assumption on the behaviour of $\mathbf{Q}$, the first point in time all registers are zero equals the first point in time the loop register attains zero. But by our definition of the iteration count, this just means that $c(\widehat{R}) = \operatorname{count}_{\mathbf{Q}}(i)$, and Lemma 11 shows that $\overline{q}(\widehat{R}) = \overline{\operatorname{tail}}^{c(\widehat{R})}(0) = c(\widehat{R}) = \operatorname{count}_{\mathbf{Q}}(i)$.

All that remains is to produce the initial register state $R_{(i)}$ with only $R_{(i)}(\mathsf{input_Q}) = i$ and $R_{(i)}(\mathsf{loop_Q}) = 1$ non-zero. For this, we define

$$r(s) = \operatorname{head}(\operatorname{tail}^{p_{\mathsf{loop_Q}}}(s)) :: r(\operatorname{proj}_{p_{\mathsf{input_Q}}}(s))$$

and utilise Lemma 1 to prove that $\overline{r}(i) = \overline{\mathrm{proj}_{p_{\mathsf{input}_\mathbf{Q}}}}^{i}(p_{\mathsf{loop}_\mathbf{Q}}) = p^{i}_{\mathsf{input}_\mathbf{Q}} \cdot p_{\mathsf{loop}_\mathbf{Q}} = \widehat{R_{(i)}}$. Defining $w(s) = r(q(s))$, we verify that $\overline{w}(i) = \overline{q}(\overline{r}(i)) = \overline{q}(\widehat{R_{(i)}}) = \mathrm{count}_\mathbf{Q}(R_{(i)})$. ◀

Unfortunately, the set of possible iteration count functions constitutes only a small part of the set of all computable functions. Intuitively, this is because even very small values can be the result of prohibitively expensive operations. However, this range can still be seen as containing Turing-complete fragments under certain encodings. This is what we exploit in the next step by shifting the role of the output register to the iteration count under a particular such encoding. The trick is to have each possible output value correspond to infinitely many iteration counts in a controlled way such that after having computed the result, by being self-aware of the current iteration count, we can consciously terminate the loop at one of these infinitely many counts, no matter how long the computation took.

▶ **Lemma 13.** *Given an* ITERATION*-program* $\mathbf{P}$*, there is an* ITERATION*-program* $\mathbf{Q}$ *such that for every input* i *natural,* $\mathbf{Q}$ *terminates if and only if* $\mathbf{P}$ *terminates, and furthermore if* $\mathbf{P}$ *terminates with output* $o \in \mathbb{N}$*, then* $\mathbf{Q}$ *terminates after exactly* $(3m+1)\cdot 3^{o+1}$ *iterations with all registers zero-valued where* $m \in \mathbb{N}$ *depends on* i*.*

Proof. Let $r_0, \ldots, r_{k-1} \in \mathbb{N}$ denote all the registers occurring in $\mathbf{Body_P}$ except for $\mathsf{output}_\mathbf{P}$ and $\mathsf{loop}_\mathbf{P}$ (but including $\mathsf{input}_\mathbf{P}$). We choose $\mathsf{loop}_\mathbf{Q}$ as a fresh natural number distinct from all previously mentioned registers. Both programs will have the same input register, i.e. $\mathsf{input}_\mathbf{Q} := \mathsf{input}_\mathbf{P}$. The output register of $\mathbf{Q}$ is irrelevant since we aim to have all registers reset at termination.

The body of $\mathbf{Q}$ is listed in Exhibit A. Note that the body of $\mathbf{P}$ is textually inserted at line 14. Register names main-phase, run-time, mod-three, swap-phase, copy also designate fresh natural numbers. To enhance readability, we used some lyrical freedom with the syntax: for example, **if** $R(\mathsf{output}_\mathbf{P}) \neq 0$ **then A else B end if** translates to $\mathrm{ifz}(\mathsf{output}_\mathbf{P}, \mathbf{B}, \mathbf{A})$. Since the program is somewhat complex, we will describe its function in great detail.

Execution of $\mathbf{Q}$, i.e. iterated execution of $\mathbf{Body_Q}$ until decrement of $\mathsf{loop}_\mathbf{Q}$, is split into two main phases, as signalled by the flag register main-phase. The first phase, when main-phase has value 0 (lines 2–22), is dedicated to simulating the original program $\mathbf{P}$ while keeping track of the total iteration count in a dedicated register run-time. At the end of this phase, after $\mathbf{P}$ has exited with result $o \in \mathbb{N}$ in register $\mathsf{output}_\mathbf{P}$, we want the total iteration count to equal $3m+1$ for some arbitrary $m \in \mathbb{N}$. The second phase, when main-phase has value 1 (lines 23–44) is dedicated to tripling the total iteration count o times, plus an additional tripling to reset run-time, so that the total iteration count becomes $(3m+1)\cdot 3^{o+1}$.

In detail, the first phase (lines 2–22) contains three separate components:

- Lines 3–4 are executed only at the beginning of the first iteration and initialise the loop register of $\mathbf{P}$ (note that the input register of $\mathbf{P}$ does not need to be initialised as $\mathsf{input}_\mathbf{P} = \mathsf{input}_\mathbf{Q}$) and mod-three (see below).
- Lines 6-12 keep track not only of the current iteration count by incrementing run-time once per iteration, but also of how many iterations modulo 3 we are afar from meeting the $(3m+1)$-condition in a dedicated register mod-three.
- Lines 13–22 execute the body of $\mathbf{P}$ once per iteration until termination is signalled by $\mathsf{loop}_\mathbf{P}$ being set to zero (lines 13–14). In subsequent iterations, the registers used in $\mathbf{P}$ are incrementally reset (lines 15–19). Finally, we wait up to two iterations for the iteration count (including the current iteration) to have the proper remainder modulo 3 (line 20), and proceed to the second phase (lines 21).

Exhibit A The body of **Q** from Lemma 13

```
 1. if $R$(main-phase) $= 0$ then
 2.     if $R$(run-time) $= 0$ then
 3.         inc(loop$_\mathbf{P}$)
 4.         inc(mod-three)
 5.     end if
 6.     inc(run-time)
 7.     if $R$(mod-three) $= 0$ then
 8.         inc(mod-three)
 9.         inc(mod-three)
10.         inc(mod-three)
11.     end if
12.     dec(mod-three)
13.     if $R$(loop$_\mathbf{P}$) $\neq 0$ then
14.         Body$_\mathbf{P}$
15.     else if $R(r_0) \neq 0$ then
16.         dec($r_0$)
17.     [...]
18.     else if $R(r_{n-1}) \neq 0$ then
19.         dec($r_{n-1}$)
20.     else if $R$(mod-three) $= 0$ then
21.         inc(main-phase)
22.     end if
23. else if $R$(swap-phase) $= 0$ then
24.     dec(run-time)
25.     inc(copy)
26.     if $R$(run-time) $= 0$ then
27.         inc(swap-phase)
28.     end if
29. else
30.     dec(copy)
31.     if $R$(output$_\mathbf{P}$) $\neq 0$ then
32.         inc(run-time)
33.         inc(run-time)
34.         inc(run-time)
35.         if $R$(copy) $= 0$ then
36.             dec(swap-phase)
37.             dec(output$_\mathbf{P}$)
38.         end if
39.     else if $R$(copy) $= 0$ then
40.         dec(swap-phase)
41.         dec(main-phase)
42.         dec(loop$_\mathbf{Q}$)
43.     end if
44. end if
```

After the final iteration of this phase, the iteration count is $3m+1$ for some $m \in \mathbb{N}$ and the only possibly non-zero registers are main-phase and swap-phase of value 1 and output$_\mathbf{P}$ of value o. We duly note that if **P** does not terminate, then neither does **Q**.

In similar detail, the second phase (lines 23–44) contains two alternately executed subphases (lines 24–28 and 30–43) responsible for shifting the iteration count back and forth between the registers run-time and copy. The current subphase is indicated by the flag register swap-phase:

- The first subphase in lines 24–28 started with register values [swap-phase : 1, run-time : $x \geq 1$, copy : 0] will end, after x iterations, with values [swap-phase : 0, run-time : 0, copy : x].
- The second subphase in lines 30–43 started with register values [swap-phase : 0, run-time : 0, copy : $x \geq 1$] will end, after x iterations, depending on the value of register output$_\mathbf{P}$,
  - if non-zero, with [swap-phase : 1, run-time : $3x$, copy : 0] and output$_\mathbf{P}$ decremented,
  - if zero, with all registers zero and loop$_\mathbf{Q}$ decremented to zero in the last iteration.

Taken together, we deduce that starting (the first subphase) with [swap-phase : 1, run-time : $x \geq 1$, copy : 0] and output$_\mathbf{P}$ non-zero, after $2x$ iterations, the effective changes will be tripling of run-time and decrement of output$_\mathbf{P}$. In particular, if x and hence run-time denoted the iteration count before these iterations, run-time will again denote the iteration count after these iterations. After following this reasoning o times, the iteration count and value of run-time will be $(3m+1) \cdot 3^o$ while output$_\mathbf{P}$ attains zero. One last instance of each phase, costing $2 \cdot (3m+1) \cdot 3^o$ iterations, yield a total iteration count of $(3m+1) \cdot 3^{o+1}$ with all registers having been cleared. ◀

4.3 Proof of the Main Result

▶ **Theorem 14.** *A unary polymorphic stream function is definable by a non-mutually recursive unary system if and only if its indexing function is computable.*

Proof. We need only consider the reverse implication. Let a computable function $\phi : \mathbb{N} \to \mathbb{N}_\perp$ be represented as an ITERATION-program $\mathbf{P}$, i.e. $\phi = \phi_P$, and let $\mathbf{Q}$ be the modified ITERATION-program as defined in Lemma 13. By Lemma 12, there is a non-mutually recursive unary system defining a stream function w such that $\overline{w}(i) = \mathrm{count}_{\mathbf{Q}}(i) = (3m+1) \cdot 3^{\phi_{\mathbf{P}}(i)+1}$ for all $i \in \mathbb{N}$ and some $m \in \mathbb{N}$ depending on i. As stated at the beginning, the latter expression is taken to mean $\perp$ if $\phi_{\mathbf{P}}(i) = \perp$.

Our strategy for extracting the final output value $\phi_{\mathbf{P}}(i)$ from this expression is by iterating a second program, adding a tail for each time the stream index is divisible by 3. In particular, from Lemma 6, it is clear how to give a non-mutually recursive unary system defining u such that

$$\overline{u}(k) = \begin{cases} k/3 & \text{if } 3 \mid k, \\ 0 & \text{else} \end{cases}$$

since this is a Collatz function. But note that we can alternatively directly define

$$\begin{aligned} u(s) = {}& \mathrm{head}(s) :: \mathrm{head}(s) :: \mathrm{head}(s) :: \\ & \mathrm{head}(\mathrm{tail}(s)) :: \mathrm{head}(s) :: \mathrm{head}(s) :: \mathrm{tail}^3(u(\mathrm{head}(s) :: \mathrm{tail}^2(s))) \end{aligned}$$

using only a single equation. For either choice, we define $v(s) = \mathrm{head}(s) :: \mathrm{tail}(u(v(\mathrm{tail}(s))))$. A second application of Lemma 11 shows that $\overline{v}((3m+1) \cdot 3^{i+1}) = i + 1$ for $i, m \in \mathbb{N}$. This function is of almost as critical importance as w as it repeats each natural number output infinitely many times in a controlled way and reverses the iteration count result encoding of program $\mathbf{Q}$.

We now have all the parts necessary for concluding our venture. Defining $f(s) = w(v(\mathrm{head}(s) :: s))$, we see that

$$\begin{aligned} \overline{f}(i) &= \max(\overline{v}(\overline{w}(i)) - 1, 0) \\ &= \max(\overline{v}((3m+1) \cdot 3^{\phi_{\mathbf{P}}(i)+1}) - 1, 0) \\ &= \max((\phi_{\mathbf{P}}(i) + 1) - 1, 0) = \phi_{\mathbf{P}}(i) \end{aligned}$$

with our usual convention regarding $\perp$, proving $\overline{f} = \phi_{\mathbf{P}}$. ◀

Further Work

A thorough inquisition of the proofs in the previous section shows that, in total, ten equations were defined for proving the main result, two of which can be inlined. It is natural to ask: what is the minimum number of unary equations required to define an arbitrary computable indexing function? The answer is *four* (and still being free of mutual recursion), but space constrains prevent us from presenting the rather involved proof.

Furthermore, we can show that recognising productivity is undecidable with complexity Π^0_2 even for unary systems with only *two* non-mutually recursive equations. Interestingly, we can prove that productivity is decidable for a *singleton* unary system. Again, the proof of this positive result is quite complex and in need of more space to be presented.

Altogether, this amounts to an exhaustive classification of definability and complexity of recognising productivity based on unary system size. However, it would be euphemistic to say that the proofs involved are not very abstract, making them even more unsuitable for exposition in a conference article.

Acknowledgements

The authors would like to thank Venanzio Capretta for introducing us to the definability problem for polymorphic stream equations and spending much time with proof checking and helpful discussions, Nicolai Kraus for proofreading, the Functional Programming Laboratory at the University of Nottingham for the encouraging research environment, and the anonymous referees for providing helpful suggestions regarding the exposition of our results.

References

1 Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Containers - constructing strictly positive types. *Theor. Comp. Sci.*, 342:3–27, 2005.

2 Corrado Böhm and Giuseppe Jacopini. Flow diagrams, Turing machines and languages with only two formation rules. *Commun. ACM*, 9(5):366–371, 1966.

3 Wilfried Buchholz. A term calculus for (co-)recursive definitions on streamlike data structures. *Ann. Pure Appl. Logic*, 136(1-2):75–90, 2005.

4 John H. Conway. Fractran: A simple universal programming language for arithmetic. In T. M. Cover and B. Gopinath, editors, *Open Problems in Communication and Computation*, chapter 2, pages 4–26. Springer, 1987.

5 Edgar W. Dijkstra. On the producitivity of recursive definitions. EWD749, 1980.

6 Jörg Endrullis, Herman Geuvers, Jacob G. Simonses, and Hans Zantema. Levels of undecidability in rewriting. *Inf. Comput.*, 209(2):227–245, 2011.

7 Jörg Endrullis, Clemens Grabmayer, and Dimitri Hendriks. Data-oblivious stream productivity. In *Proc. 15th Int. Conf. on LPAR*, LPAR '08, pages 79–96. Springer, 2008.

8 Jörg Endrullis, Clemens Grabmayer, and Dimitri Hendriks. Complexity of Fractran and productivity. In *CADE*, pages 371–387, 2009.

9 Jörg Endrullis, Clemens Grabmayer, Dimitri Hendriks, Ariya Isihara, and Jan Willem Klop. Productivity of stream definitions. In *Proc. FCT 2007*, volume 4639 of *LNCS*, pages 274–287. Springer, 2007.

10 Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D. Lawson, Michael Mislove, and Dana S. Scott. *Continuous Lattices and Domains*, volume 93 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2003.

11 David Harel. On folk theorems. *SIGACT News*, 12:68–80, 1980.

12 Andreas M. Hinz. The Tower of Hanoi. *Enseign. Math.*, 35(2):289–321, 1989.

13 Stephen C. Kleene. Recursive predicates and quantifiers. *Trans. AMS*, 53(1):41–73, 1943.

14 Stuart A. Kurtz and Janos Simon. The undecidability of the generalized Collatz problem. In *TAMS*, volume 4484 of *LNCS*, pages 542–553. Springer, 2007.

15 Jeffery C. Lagarias. *The Ultimate Challenge: The $3x + 1$ Problem.* AMS, 2010.

16 Marvin L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967.

17 Eleonora Perkowska. Theorem on the normal form of a program. *Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys.*, 22(4):439–442, 1974.

18 Grigore Roşu. Equality of streams is a Π_2^0-complete problem. In *ICFP*. ACM, 2006.

19 Jan M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theor. Comp. Sci.*, 308(1-3):1–53, 2003.

20 Jakob Grue Simonsen. The Π_2^0-completeness of most of the properties of rewriting systems you care about (and productivity). In *Proc. 20th Int. Conf. on RTA*, RTA '09, pages 335–349. Springer, 2009.

21 Robert I. Soare. *Recursively Enumerable Sets and Degrees.* Perspectives in Mathematical Logic. Springer, 1987.

22 Hans Zantema. Well-definedness of streams by transformation and termination. *LMCS*, 6(3), 2010. paper 21.

Matching of Compressed Patterns with Character-Variables*

Manfred Schmidt-Schauß[1]

1 Goethe-Universität, Frankfurt, Germany
schauss@ki.informatik.uni-frankfurt.de

Abstract

We consider the problem of finding an instance of a string-pattern s in a given string under compression by straight line programs (SLP). The variables of the string pattern can be instantiated by single characters. This is a generalisation of the fully compressed pattern match, which is the task of finding a compressed string in another compressed string, which is known to have a polynomial time algorithm. We mainly investigate patterns s that are linear in the variables, i.e. variables occur at most once in s, also known as partial words. We show that fully compressed pattern matching with linear patterns can be performed in polynomial time. A polynomial-sized representation of all matches and all substitutions is also computed. Also, a related algorithm is given that computes all periods of a compressed linear pattern in polynomial time. A technical key result on the structure of partial words shows that an overlap of $h + 2$ copies of a partial word w with at most h holes implies that w is strongly periodic.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases matching, grammar-based compression, partial words

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.272

Category Regular Research Paper

1 Introduction

Matching is a widely used operation in computer science: Given a string (term, object)-pattern s with variables, and a string (term, object) t, is there a substitution σ, such that $\sigma(s)$ is a substring (subterm, subobject) of t? There are at least two sources of motivation for considering matching problems: (i) Given a term rewrite system R and a term t, in order to rewrite t it is first necessary to find a substitution σ, and a rule $s \to r$ of R, such that $\sigma(s)$ is a subterm of t. (ii) Given a string t or a data base D, and a pattern s, find and retrieve the substrings of t or the records (objects) in D, respectively, that are matched by s. Here the pattern may be a string, a string with variables, with holes, or a regular expression, or otherwise extended patterns. The task is to design efficient algorithms for the different variants of matching. In the following we sometimes speak of *submatching* in order to emphasize that substrings (subterms) are searched.
Often terms, strings or databases are very large, and are thus represented or stored in a compressed format. This is the motivation to propose and analyze efficient variants of matching, term rewriting or other algorithms that can also be run on the compressed strings

* Partially supported by DFG, grant SCHM 986/9-1

or terms. We do not directly tackle the full problem of compressed term-submatching. The reason is that this turns out to be a nontrivial problem with complex algorithms even for specializations if polynomial-time algorithms are requested: The technical report [24] contains an approach to compressed term submatching and compressed rewriting but has to leave open several issues. In particular wether there is a polynomial-time algorithm for finding a redex for rewriting where the rule as well as the to-be-rewritten term are compressed. Even the complexity-question of term-submatching for left-linear rules was left unsolved. Thus this paper is restricted to matching algorithms on compressed strings where the patterns are linear and variables mainly stand for characters. The result in this paper turns out to be a first step in the construction of a polynomial-time algorithm for finding matches of compressed left-linear rules in compressed terms. Note that compressed (exact) term matching, where s should match the whole term t is known to be polynomial [11].

Potential applications of matching are in information retrieval, in optimizing term rewriting systems on large terms, and in application domains where large strings are processed. Citing the book [6], where algorithms on partial words (string patterns which are linear in the variables) are investigated: "The next generation of research on combinatorics of partial words promises to have a substantial impact on molecular biology, nanotechnology, data communication, and DNA computing". Our algorithm has potential applications in finding DNA-fragments under compression taking care of SNPs (single nucleotide polymorphism), which is a special case of a point mutation.

Well-known compression methods and compressed representations of strings are the LZ77, LZ78 schemes [26, 27], where LZ77 may compress a string of length n into a representation of $O(\log(n))$ size. Other compression mechanisms are straight line programs (SLP), which are context free grammars (in Chomsky normal form) that generate exactly one string. It is well-known that there are efficient translations between LZ77 and SLPs. A prominent result is that SLP-compressed strings can be tested for equality in polynomial time in the size of SLPs using the method of Plandowski [22] (with improvements in [17, 12]). Plandowski's algorithm works top-down in the grammar and keeps the space (and time) requirements small by exploiting periodicities to detect and eliminate redundancies, whereas Lifshits' is a dynamic programming algorithm that memorizes periodicities using arithmetic progressions. Fully compressed matching of compressed strings (finding a string as a substring in another string) can also be decided and computed in polynomial time [14]. An efficient algorithm is given by Lifshits [17], who shows that it can be done in time $O(mn^2)$ where m is the size of the SLP for the pattern and n is the size of the SLP of the string. An algorithm with improved complexity is proposed by Jez [12]. Complexities of further algorithms on SLP-compressed strings are in [18]. There are also generalizations of SLP-compression on two-dimensional texts, see [2]. The SLP-compression scheme is also extended to terms and ranked trees: An application and an analysis for XML-compressions is in [9], and an analysis and application to matching and unification is in [11].

The matching problem for strings is the question, given a pattern string s, whether it occurs in another given (long) string t. Well-known and folklore algorithms scanning the string t and comparing it with s are for example the Knuth-Morris-Pratt and the Boyer-Moore algorithm. An example for a generalization is searching for an instance of a regular expression in a string, which is possible by a polynomial-time algorithm. There are also results for regular string matching in compressed strings, which is also known to be executable in polynomial time [4, 21], where however, the compression schemes LZ78 or LZW ([26, 27]) are used that have a maximum compression ratio of $O(\sqrt{n})$, in contrast to straight line programs (SLP) which have a potential compression ratio up to $O(\log n)$. In [19] the complexity of several variants

of the matching problem of regular expressions in SLP-compressed texts is investigated: for the usual regular expressions with concatenation, union, and star-operator, the matching problem is P-complete [20], whereas for slightly extended regular expressions including integer exponents, the matching problem in an SLP-compressed string is PSPACE-hard [19].
The string-matching problem when both strings s, t are SLP-compressed is called fully compressed matching. It is well known that the fully compressed pattern matching problem can be decided in polynomial time [17, 12]. It is also well-known that if s contains string-variables and s, t are both uncompressed, then exact matching is NP-complete [1]. The same problem restricted to the case where s is linear, but where the pattern may be compressed, can easily be decided in polynomial time by an eager search (see Lemma 5.1). A related paper is [25], where the pattern s is linear, uncompressed and may contain character- and string-variables and where t is SLP-compressed, matching is shown to be in $PTIME$.
In this paper we consider the following string matching problem: The pattern s is SLP-compressed by G_s, s may have at most $O(|G|)$ holes, and t is also a string SLP-compressed by G_t. We use the view that the holes are character-variables (that may be instantiated with characters from the signature). The question is whether there exists a substitution σ (replacing variables by characters), such that $\sigma(s)$ is a substring of t. The methods as mentioned above lead to exponential time algorithms, so a different algorithm is required. As a technical sub-problem we have to analyze structural properties of strings with character-variables, where every variable occurs at most once. Such strings are more or less equivalent to strings with holes (where a hole means a missing or unknown character). This notion is equivalent to *partial words*. Some results on the structure of partial words and simple unification equations on partial words are in [6, 7].

As result (Algorithm 4.1, Theorem 4.5) we describe an algorithm that runs in polynomial time, decides this problem and moreover outputs a polynomial representation of all occurrences of the pattern. A technical result that is required to show polynomiality of the algorithm is a periodicity-lemma (Theorem 3.10): Given a partial word w with h holes, and an overlap of $h + 2$ copies of w where the occurrences are dense enough, we show that w is strongly periodic, together with a formula for the period. As a corollary, we show that if a partial word has two periods p, q, then it has $\gcd(p, q)$ as periods, provided further conditions on p, q, the length $|w|$ and the number of holes hold. We also show that there is a polynomial time algorithm to compute all periods of a compressed partial word (Proposition 4.6).
Also extensions are considered: The algorithm can be extended to the case where s contains character and string-variables, and s contains every variable at most once (Proposition 5.2). Another extension is that the variables in the pattern s may occur several times, and s is compressed. Then a modification of the algorithm runs in polynomial time in the number of variable occurrences, and the size of the SLPs (Proposition 5.3).
The paper is structured as follows. After some preliminaries on string properties and compression, in Section 3 words with holes, periodicity and overlaps of a word with holes with itself are analyzed. In Section 4 a dynamic programming algorithm is described that computes all matching occurrences of one partial word in a string, both compressed. Finally, some potential extensions are discussed in Section 5.

2 On Strings and Compression

We provide some preliminaries for our algorithm and results: strings, overlaps and periodicity, straight line programs (SLP) for compressing strings and several algorithms and operations on SLPs.

2.1 Properties of Strings

If s is a string (also called a word) over a finite alphabet Σ of characters, we write $s[i]$ for the symbol of s at position i, where we assume that $s = s[0] \dots s[n-1]$, if $n = |s|$, where $|s|$ denotes the length of s. The substring of s from positions i to j is denoted as $s[i..j]$. We also use t^k for a k-fold concatenation of t with $k \geq 0$, where $t^0 = \varepsilon$ is the empty string.
If for a string s there is a number $1 \leq p < |s|$ such that $s[i] = s[i+p]$ for all i, provided $s[i], s[i+p]$ are defined, then we say that s is *periodic with period* p. For strings this is equivalent to: for all i, j: $i \equiv j \texttt{ mod } p$ implies $s[i] = s[j]$ for all $0 \leq i, j \leq |s| - 1$.

▶ **Theorem 2.1.** *(Fine and Wilf [10]) Let s be a word with periods p and q. If $|s| \geq p + q - gcd(p,q)$, then s has a period $gcd(p,q)$.*

A string w is a *cyclic permutation* of another string w', iff there are substrings w_1, w_2 of w, such that $w = w_1 w_2$, and $w' = w_2 w_1$.

▶ **Lemma 2.2.** *If w is a p-periodic string, then any two substrings v, v' of w of length $|v| = |v'| = p$ are cyclic permutations of each other.*

The following is easy and can be proved by induction using an Euclidean algorithm.

▶ **Fact 2.3.** *If for two nontrivial strings s_1, s_2, the equation $s_1 s_2 = s_2 s_1$ holds, then there is a nontrivial string t, and natural numbers $m, n \geq 1$, such that $s_1 = t^n, s_2 = t^m$. Thus, $|t|$ is a period of $s_1 s_2$ and also a divisor of $gcd(|s_1|, |s_2|)$.*

▶ **Fact 2.4.** *Let w be a string that overlaps with itself, i.e. $w = w_1 w_2$, and w_2 is a prefix of w. Then w is periodic with a period $|w_1|$.*

We will also use strings with variables. Let us assume that there are two kinds of variables: (i) variables standing for characters and (ii) variables standing for strings. We assume that substitutions respect the kind of variables. In the following, variables are character-variables, if not stated otherwise.

2.2 Compression of Strings and Straight Line Programs

▶ **Definition 2.5.** A *straight-line program (SLP) G* (see [22, 23, 13]) is a context free grammar with the following restrictions: Every nonterminal generates exactly one string, for every nonterminal there is exactly one rule of one of the forms $A \to a$ for $a \in \Sigma$ or $A \to A_1 A_2$, where A, A_1, A_2 are nonterminals, and the grammar is not recursive. The generated string for nonterminal A is denoted as $val(A)$, and the string generated by the start nonterminal is denoted as $val(G)$. The *size* of G is the number of its rules, and the *depth* of G is the depth of the derivation tree of the start nonterminal.

Note that $|val(G)|$ may be as large as $2^{|G|}$, but not larger. Lempel-Ziv compression schemes can be efficiently translated into SLPs [23]. However, SLP are more amenable to general analyses and proofs about efficient algorithms for compressed strings.

▶ **Lemma 2.6.** *(Plandowski [22]) Given two nonterminals A, B of an SLP G, it is decidable in polynomially time in $|G|$, whether $val(A) = val(B)$, where an efficient $O(|G|^3)$-algorithm was proposed in [17], and the recently proposed matching algorithm by Jez [12] further improves the complexity.*

The following operations, extensions and modifications of an SLP can be done efficiently (see [15, 16, 11]).

▶ **Lemma 2.7.** *Let G be an SLP. Then a sequence of m operations can be done in polynomial time $O(poly(|G|, m))$, where poly is a polynomial and the operations may be the following:*

1. *Given a nonterminal A, and a prefix (or suffix) w of $val(A)$, extend the SLP to G' such that there is a nonterminal B with $val(B) = w$. In this case the depth of the SPL G' has the same depth as G.*
2. *Given nonterminals A_1, A_2, extend the SLP to G' by $B \to A_1A_2$, where B is a fresh nonterminal.*
3. *Given a nonterminal A, and a number $n \leq 2^{|G|}$, extend the SLP to G' such that there is a nonterminal B with $val(B) = val(A)^n$.*

Note that the complexity is not just $m * poly'(|G|)$, since the extensions iteratively increase the size of intermediate grammars G.
Mainly based on Plandowski's result [22], the following holds:

▶ **Lemma 2.8.** *Let G be an SLP. Then the following can be performed in polynomial time:*

1. *Given a nonterminal A, compute the length $|val(A)|$.*
2. *Given two nonterminals A, B, check whether $val(A)$ is a prefix (suffix) of $val(B)$.*

2.3 Equality of Compressed Strings

▶ **Lemma 2.9.** *Equality up to renaming of variables of two compressed strings with variables can be tested in polynomial time.*

Proof. Let A, B be the nonterminals that represent the two strings that are to be compared, where we assume for simplicity that there are no nonterminals shared in the derivation of A and B. First we normalize the variable names and then we apply a compressed-equality test. The normalization is top down in the grammar: For every variable x identify its leftmost occurrence in $val(A)$, and then replace it by y_i, where i is the number of different variables occurring to the left of this occurrence in $val(A)$, and where y_i is a unique name that is used by the normalization. The y_i can be seen as variables or as constants not occurring in $val(A) \cup val(B)$. Then do the same for B. The number of variables is at most $|G|$, and for every variable the normalization can be done in polynomial time. Note that there is no size-change of G. Then use the equality-test for compressed strings. ◀

▶ **Lemma 2.10.** *Let s be a string with character variables and t be a string, both compressed with an SLP G. Then it is decidable in polynomial time whether there exists σ, such that $\sigma(s) = t$.*

Proof. Let there be $n < |G|$ variables in s. It is sufficient to compute a single position p_i (say the leftmost one) for every variable x_i in s for all $i = 1, \ldots, n$. Then compute the character a_i at position p_i of t. and perform the equality test of $[a_1/x_1, \ldots, a_n/x_n]s = t$. ◀

2.4 Submatching

▶ **Definition 2.11.** The *compressed character-submatch (CCSM)* problem for strings is defined as follows. Let Σ be an alphabet, let G be an SLP over Σ, let $t = val(T)$ where T is a nonterminal of G, let $s = val(G')$ where G' is an SLP over $\Sigma \cup V$, where V is a set of (character-)variable symbols.
Question: is there a substitution $\sigma : V \to \Sigma$ such that $\sigma(s)$ is a substring of t? If every variable occurs at most once in s, then it is called the *linear compressed character submatch (LCCSM)* problem.

Note that if σ can replace variables by strings, then even the following question is NP-complete: Given uncompressed s, t, does there exists σ, such that $\sigma(s) = t$?
The following observations for special cases are easy to verify:

- CSSM is in NP: guess the substitution σ and then use fully compressed pattern matching.
- If there is only one variable x, then CSSM is in *PTIME*: try all possibilities for σ.
- If there is a given number k that is an upper bound for $|val(s)|$, then CSSM is solvable in polynomial time where k occurs in the exponent of the polynomial.

2.5 Remarks on Comparison with Other Approaches and Result

approximate matching with edit-distance. This is the question whether a variant s' of the string s occurs in t, where s' and s differ by a form of edit-distance. This is also called approximate matching. The investigations in the literature usually take a weaker form of compression: LZ78, LZW, and the pattern s is not compressed. If s is not compressed, then the approximate matching will also produce the matching solutions in case s has character holes, but perhaps more, depending on the exact definition of edit-distance used [4].

regular expression matching. Compression by SLP cannot be represented using a regular expression. If the pattern has the form $s = s_1x_1 \dots x_ns_{n+1}$ and x_i are character-variables or string variables, and s_i are uncompressed and ground (no variables), then this can be translated into a regular expression match question. If the regular expression syntax also allows exponents w^k for an integer k and a single string w, then it can also cover special forms of compression, but not the general form. However, this syntax extension is not discussed in [19]. It is not hard to see that in this case matching in SLP-compressed strings is NP-hard (using NP-hardness of SUBSETSUM and exploiting the union-operator).

3 The Linear Case: Partial Words

We mainly consider the linear compressed character submatch problem (LCCSM). A string where every variable occurs at most once can also be seen as a string with several holes where the holes represent single missing or unknown characters. We denote the hole with $\circ$, or with variable names x, y, z, depending on the context. In the literature these are also called *partial words*, see for example [8, 5, 6].
If for a partial word s there is a number $p < |s|$ such that $s[i] = s[i+p]$ for all i, and $s[i], s[i+p]$ are defined and are not holes, then we say that s is *p-locally periodic* with period p. For example, the partial word $ababa{\circ}acac$ is 2-locally periodic. If there is a number $p < |s|$ such that $i \equiv j \ \mathtt{mod}\ p$ implies $s[i] = s[j]$ for all $0 \leq i, j \leq |s| - 1$ if $s[i], s[j]$ are defined and not holes, then s is called *periodic* (also strongly periodic), and p is a *period* of s. The partial word $ab{\circ}bababa$ is 2-periodic. Note that p-periodic implies p-locally periodic, but the converse might not hold, for example $ab{\circ}bb$ is 2-locally periodic, but not 2-periodic. Note that p-periodic partial words can be made p-periodic strings by substituting characters into the holes, whereas this is not the case for p-locally periodic partial words. We write $s =_h s'$ for two partial words s, s' if $|s| = |s'|$ and s, s' are equal up to the occurrences of holes. Note that $=_h$ is not transitive, for example $a{\circ} =_h {\circ}b$, ${\circ}b =_h b{\circ}$, but $a{\circ} \neq_h b{\circ}$.
Equality $=_h$ of two partial words w, w' that are SLP-compressed, and if holes are represented as variables, can be checked polynomially in the size of the SLP and the number of holes of w, w'. If holes were an extra symbol in the SLP, and there is no restriction on the number of occurrences, then we could not find a polynomial algorithm to decide $=_h$: adapting the

Plandowski-style algorithms is not possible, since periodicity lemmas fail in this case. The only trivial information is that the problem is in co-NP.

3.1 Partial Words with One Hole

First we investigate some properties of (uncompressed) partial words with one hole. In the following we fix an alphabet Σ. The inner structure of a p-locally periodic string is clarified:

▶ **Lemma 3.1.** *Let s be a partial word with one hole at position r, and let s be p-locally periodic. Then s is either p-periodic, or the following holds:*

- *For all $i, j : i \equiv j \bmod p$ and $i \not\equiv r \bmod p$ it holds that $s[i] = s[j]$, if $s[i], s[j]$ is defined;*
- *$s[0..r]$ as well as $s[r..|s| - 1]$ are p-periodic.*

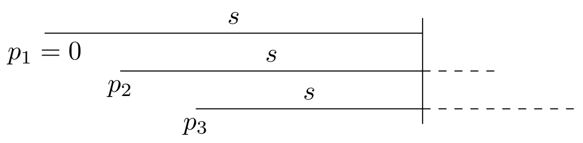
An example for such a partial word is $ababa\circ acac$, which is 2-locally periodic, but not 2-periodic, where $\circ$ represents the hole.
There are generalizations of the Fine and Wilf-theorem to partial words:

▶ **Theorem 3.2.** *([3, 8]) Let s be a partial word with one hole. If s is p-locally periodic and q-locally periodic, $p \neq q$, and $|s| \geq p + q$, then s is also $gcd(p, q)$-periodic.*

The bound $|s| \geq p + q$ is sharp [3, 8].

▶ **Definition 3.3.** Let s be a partial word and $n \geq 2$. An n-fold *overlap* of s (with itself) is given by starting positions $0 \leq p_1 < p_2 < p_3 < \ldots < p_n \leq |s| - 1$, such that for all $i, j = 1, \ldots, n$ and $0 \leq k \leq |s| - 1$: if $0 \leq k - p_i, 0 \leq k - p_j$, then $s[k - p_i] = s[k - p_j]$, provided neither $s[k - p_i]$ nor $s[k - p_j]$ is a hole.
In the following we let $p_1 = 0$, if not stated otherwise.

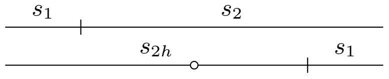
The overlap of s with itself intuitively is only meant in the range of the left most occurrence, left of the vertical line. There are no overlap conditions right of the vertical line. The reason for this choice is our application to overlapping prefixes below. For example, for a 3-overlap, equality of characters must hold on every position of the topmost occurrence of s.

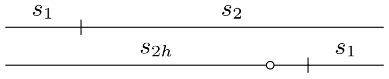


A simple example of an overlap is $s = ab\circ baba$, $n = 2$ and $p_2 = 2$. Another overlap is $s = a\circ bbb$, where $n = 2$ and $p_2 = 1$.
The following shows that the periodicity claim in Fact 2.3 also holds if one of s_1, s_2 is a partial word with one hole, and the other one is a word without holes.

▶ **Lemma 3.4.** *Let s_1, s_2 be nontrivial strings, and let $s_{i,h}, i \in \{1, 2\}$ be partial words with a single hole such that $s_{i,h} =_h s_i, i \in \{1, 2\}$. If $s_1 s_2 =_h s_{2,h} s_1$, or if $s_1 s_2 =_h s_2 s_{1,h}$, then $s_1 s_2$ is periodic with a period $gcd(|s_1|, |s_2|)$.*

Proof. We use induction on $|s_1 s_2|$. The base case is $|s_1| = |s_2|$. In this case each equation implies that $s_1 = s_2$, and the period has length $|s_1| = gcd(|s_1|, |s_2|)$.
Let $|s_1| \neq |s_2|$. We first consider the case $s_1 s_2 =_h s_{2,h} s_1$, the other case is symmetric. We distinguish several cases:





- $|s_2| > |s_1|$ and the hole is at a position $\leq |s_2| - |s_1|$.
  Then let s'_2 be such that $s'_2 s_1 = s_2$. Let $s'_{2,h}$ such that $s'_{2,h} s_1 := s_{2,h}$. The induction hypothesis can be applied to $s_1 s'_2 =_h s'_{2,h} s_1$, and shows that a period has length $gcd(|s_1|, |s_2|) = gcd(|s_1|, |s'_2|)$, and it is also the period of $s_1 s_2$.
- $|s_2| > |s_1|$ and the hole is at a position $\geq |s_2| - |s_1|$.
  Then let s'_2 be such that $s'_2 s_1 = s_2$. Let $s'_{1,h}$ be such that $s_{2,h} =_h s'_2 s'_{1,h}$. The induction hypothesis can be applied to $s_1 s'_2 =_h s'_2 s'_{1,h}$, and shows that a period has length $gcd(|s_1|, |s_2|) = gcd(|s_1|, |s'_2|)$, and it is also the period of $s_1 s_2$.
- $|s_1| > |s_2|$. This can be proved similarly as above where only the occurrence of the hole is in different partial words. ◀

Lemma 3.4 and Theorem 3.2 imply:

▶ **Corollary 3.5.** *Let s_1, s_2 be nontrivial strings, such that $s_1 s_2$ is periodic with period p, $p \leq 0.5|s_1 s_2|$, and let $s_{i,h}, i \in \{1, 2\}$ be partial words with a single hole such that $s_{i,h} =_h s_i, i = 1, 2$. If $s_1 s_2 =_h s_{2,h} s_1$ or $s_1 s_2 =_h s_2 s_{1,h}$, then $s_1 s_2$ is periodic with a period $gcd(|s_1|, |s_2|, p)$.*

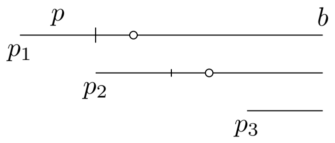
▶ **Lemma 3.6.** *Let w be a partial word with one hole, and assume that there is a 2-overlap of w, starting at p_1, p_2. Then w is locally periodic with period $p_2 - p_1$.*

Proof. The overlap immediately implies that w is locally periodic with period $p_2 - p_1$. Lemma 3.1 provides information about the exact structure of w. ◀

The following lemma shows a part of Lemma 3.9, which is the the base case of the induction proof in Theorem 3.10.

▶ **Lemma 3.7.** *Let w be a partial word with one hole, and assume that we have a 3-overlap of w, starting at $0 = p_1 < p_2 < p_3$. Let $p = p_2 - p_1$ and assume that $|w| - p_3 \geq p$, i.e., the last p positions are common to all the three occurrences of w. Then the partial word w is periodic, and a period is $gcd(p_2 - p_1, p_3 - p_2)$. Moreover, the overlap is consistent with using the same substitution for the single variable for every occurrence of w.*

Proof.



For the occurrences starting at p_1, p_2, Lemma 3.6 implies that w is locally periodic with period $p = p_2 - p_1 = p_2$. Also, if t_1, t_2 are substrings of length p of w, then either t_1, t_2 are cyclic permutations of each other, or cyclic permutations up to the occurrence of one character. Note that the overlap does not imply that w is $(p_3 - p_2)$-locally periodic, since there is a cut at b, hence Theorem 3.2 is not applicable.

- Let $b := |w| - 1$. If the third occurrence does not contain a hole in the overlap, i.e. $w[0..b - p_3]$ does not contain a hole, then the string $w[b - p_3 - p + 1..b - p_3]$ is equal to a substring of length p of the first or of the second occurrence, without hole. This equality implies that there are two equal substrings t'_1, t'_2 of w, both of length p, where t'_1 is left of the hole and t'_2 is right of the hole. By Lemma 3.6, this is only possible if w is periodic with period p.
- If the third occurrence contains a hole in the overlap, then there are two subcases: (i) $r \geq p$, where r is the hole position in w. Then let t_1 be the substring $a[r - p_3 - p..r - p_3 - 1]$, which is the substring of the third occurrence immediately left to the hole. Due to the overlap this is equal to a substring t_1 of length p in the first occurrence of w, which is a substring in w right of the hole. Again, this implies that w is periodic with period p.
  (ii) $r < p$, i.e. the hole is in the first p-period of w. Then w is periodic with period p.

Now it remains to show that also $p_3 - p_2$ is a period of w. If p is a divisor of $p_3 - p_2$, then there is nothing to show, so let us assume that p is not a divisor of $p_3 - p_2$. Since w is already periodic with period p, it is sufficient to show that a substring of w of length p is also periodic with period $p_3 - p_2$, and then apply Theorem 3.2. The prefix of the third occurrence, i.e. $w[0..p-1]$ overlaps a part of the first occurrence: $w[p_3..p_3+p-1]$. Either $w[0..p-1]$ contains a hole and $w[p_3..p_3+p-1]$ does not contain a hole, or vice versa. Moreover, due to p-periodicity of w, there are nontrivial s_1, s_2, s'_1, s'_2 such that $s_1s_2 = w[0..p-1]$ and $s'_2s'_1 = w[p_3..p_3+p-1]$ where $|s_1| = |s'_1|$, $|s_2| = |s'_2|$, and $|s_2| \equiv p_3 - p_2 \mod p$. Also, depending on where the hole is, we have either

- $s_1s_2 = s'_2s'_1$, $s_1 = s'_1$, $s_2 = s'_2$, or
- $s_1s_2 =_h s'_2s'_1$, $s_1 = s'_1$, $s_2 =_h s'_2$, or
- $s_1s_2 =_h s'_2s'_1$, $s_1 =_h s'_1$, $s_2 = s'_2$

Then Fact 2.3 shows the claim on periodicity in the first case, and Lemma 3.4 shows the claim on periodicity in the other cases. ◀

▶ **Remark 3.8.** *The number of common overlap positions in Lemma 3.7 is sharp. For example, let $w = bab\circ bcb$, and let $p_2 = 2, p_3 = 6$ with $|w| = 7$ be an overlap. Then only one character position is common to all the three occurrences. The string $bab\circ bcb$ is not 2-periodic.*

The following lemma is the base case of the induction proof in Theorem 3.10.

▶ **Lemma 3.9.** *Let w be a partial word with one hole, and assume that we have an overlap of $k \geq 3$ occurrences of w starting at $0 = p_1 < p_2 < \ldots < p_k$, respectively. Let $p = p_2 - p_1$ and assume that the last p positions, i.e., $|w| - p, \ldots, |w| - 1$ are common to all the k occurrences. Then the partial word w is periodic, and a period is $gcd(p_2 - p_1, p_3 - p_2, \ldots, p_k - p_{k-1})$.*

Proof. This follows by a repeated application of Lemma 3.7 and Corollary 3.5 as follows, by induction on the number n of overlaps:
Assume that for $j \geq 3$ it is shown that the period is $gcd(p_2 - p_1, p_3 - p_2, \ldots, p_j - p_{j-1})$. Then consider the next occurrence of the partial word w, and focus on the overlap of the occurrence $j + 1$ of w at positions $|w| - p, \ldots, |w| - 1$ with the first or second occurrence of w. We take the first one, if there is no hole in the interval $w[|w| - p..|w| - 1]$, otherwise we take the second one. Due to periodicity with period p, this leads to one of the equations $s_1s_2 = s_2s_1$, or $s_1s_2 =_h s_{2,h}s_1$ or $s_1s_2 =_h s_2s_{1,h}$ with $s_i =_h s_{i,h}$ for $i = 1, 2$, and $|s_1| \equiv p_j \mod p$. Simple computation with gcd now shows the claim. ◀
Now we are in a position to perform an induction proof for overlaps of any number of occurrences of a partial word with several holes.

▶ **Theorem 3.10.** *Let w be a partial word with n holes, and assume there is an overlap of $m \geq n + 2$ occurrences of w, starting at $p_1 < p_2 \ldots < p_m$. Let p_{max} be $\max\{p_{i+1} - p_i \mid i = 1, \ldots, m-1\}$, and $|w| - p_m \geq 2n \cdot p_{max}$; this means there are $2n \cdot p_{max}$ common positions of all occurrences of w.*
Then the partial word w is periodic, and a period is $p_{all} := gcd(p_2 - p_1, p_3 - p_2, \ldots, p_m - p_{m-1})$. There is a unique string w' that can be overlapped with w, starting at m ending at $|w| - 1$, that is periodic with the same period. Moreover, the overlap is consistent with using the same substitution for every occurrence of w.

Proof. By induction on the number of holes. If w has one hole (i.e. $n = 1$), and there are $m \geq n + 2 = 3$ occurrences, then the claim follows from Lemma 3.9.

Now assume that there are $n > 1$ holes in w. Let r be the position of the leftmost hole of w, and let w' be a partial word, v_1 be a string, such that $w = v_1 \circ w'$. There are two cases:

1. $r < 2p_{max}$. The overlap of $m \geq n+2$ occurrences of w' is constructed from the given overlap by cutting away all prefixes of length $r+1$. For convenience, we have different starting positions $p'_i := p_i + r + 1$. The number of holes is decreased by one, the maximum p_{max} is unchanged, and the condition $|w'| - p_m + 1 \geq 2n' \cdot p_{max}$ with $n' = (n-1)$ holds. Now we can use induction and obtain that w' is periodic with the period as claimed. It remains to show that this claim can be transferred to the full partial word w: We look at the prefix w_1 of the m^{th} (the last) occurrence of w. This string w_1 is positioned such that there are at least $n+1$ other occurrences of w with a common position. For the position $q = p_m + r - 1$ in the prefix v_1 of the m^{th} occurrence of w, there is at least one index $j(q)$, such that $w_{j(q)}[q - p_{j(q)+1}]$ is not a hole. Thus this character is determined by the overlap. We scan the positions from greater indices to smaller ones and observe that for one period p_{all} the suffix $w_{1,1}$ of w_1 is determined. It is also obvious that the partial word $w_{1,1} \circ w'$ is p_{all}-periodic. Now we repeat this process of determining w_1 period-wise and finally obtain that w is p_{all}-periodic.
2. $r \geq 2p_{max}$. This implies $|v_1| \geq 2p_{max}$. The length condition shows that in the overlap of the m occurrences, there are overlaps of v_1 with itself for all the indices $i, i, +1$ where the shift in the overlap is $p_{i+1} - p_i$, and the overlapping part is longer than $|v_1|/2 \geq p_{max}$. Fact 2.4 shows that v_1 is periodic with period $p_{i+1} - p_i$ for all i, and since $|v_1|/2 \geq p_{i+1} - p_i$ for all i, we obtain that v_1 has period p_{all} by Theorem 2.1. Since the copies of v_1 cover the overlap for all positions $0..p_m + |v_1| - 1$, we have already periodicity of $w[0..p_m + |v_1| - 1]$ with period p_{all}. Similar as for the first item, we obtain also for all further positions of w, that these satisfy the periodicity condition by iterating the following step: determine the next p_{all} characters of w using the observation that there is an overlap of at least $n+2$ occurrences and that the focussed positions are in the part where all occurrences overlap. Since at least one occurrence has a character at the questioned position, we see that the next p_{all} characters are determined and satisfy the period condition. Iterating this implies that the periodicity condition holds for w.

Since $m \geq n+2$, the string starting at m to $|w| - 1$ that overlaps all occurrences of w is uniquely defined and is periodic with the same period due to the previous arguments. The periodicity of w, and since w contains hole-free substrings of lengths greater than the period implies that the substitutions can be chosen consistently for all occurrences. ◀

▶ **Corollary 3.11.** *Let w be a partial word with $n \geq 2$ holes, let w be periodic with periods $p, q \leq |w|/(3n)$. Then w is also $\gcd(p,q)$-periodic.*

Proof. The periodicity assumptions imply that there is an overlap of w with $p_i = ip$ as well as $p_i = iq$ for $i = 0, 1, 2, \ldots$. The upper bound permits to apply Theorem 3.10: Assume $p < q$ and that q is not a multiple of p. Then in the interval $0..|w|/(3n)$ at least $n+2$ different copies of w are starting. The prerequisite of the theorem holds: $|w| - np \geq 2n \cdot p$. The derived period is $\gcd(p, q-p)$, which is the same as $\gcd(p,q)$. ◀

The corollary is too weak for $n = 1$, since Theorem 3.2 provides a better bound, which indicates that there is space for improvements in the case $n \geq 2$.

4 A Matching-Algorithm Using Dynamic Programming

Let G_s, G_t be SLPs for s and t, respectively, where s is a partial word with n holes, and t is a string, and S is a nonterminal with $val(S) = s$. First we build two tables, a *prefix-table* and a *result-table* as follows: The coordinate is T, for the nonterminals of G_t. Entries in the lists may be (i) positions a, (ii) arithmetic progressions, represented by triples (a, b, m), where a is the start position, b is the step length, and m is the number of steps, (iii) arithmetic sequences without upper bound, represented by pairs (a, b). The result table contains a list with entries a or (a, b, m), whereas the prefix table contains a list with entries a or (a, b). These entries have the following semantics:

- a represents that a prefix of $val(S)$ is a suffix of $val(T)$ where the suffix starts at position a in $val(T)$.
- (a, b, m) represents prefixes of $val(S)$ as suffixes of $val(T)$, where the starts of the S-suffixes are at positions $a + ib$ for $i = 0, \ldots, m - 1$ in $val(T)$.
- (a, b) is the same as (a, b, m) for the maximal possible m. Moreover, $val(T)$ has a periodic suffix: $val(T)[a..|val(T)| - 1]$ is periodic with period b. The period string is $val(T)[a, a + b - 1]$.

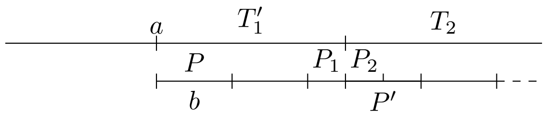
▶ **Algorithm 4.1.** *The construction of the tables is as follows and works bottom-up in G_t, where we fix S. For a nonterminal T with rule $T \to T_1T_2$, we assume that the prefix tables for T_1 and T_2 are already constructed. Then the prefix table of T will be constructed using the entries in the prefix tables of T_1 and T_2. If the entries for T are constructed, then a compaction step is applied to all entries for T. The result-table is like an output and will not be used for further constructions. For every nonterminal T, the following is performed:*

Simple cases:

1. *If $|val(T)| = 1$: If $|val(S)| > 1$, and $val(S)$ has $val(T)$ as prefix, or if $val(S)$ starts with a hole, then there is an entry 0 in the prefix-table of T, otherwise there is no entry in the prefix table. If $|val(S)| = 1$, and $val(S)$ has $val(T)$ as prefix, or if $val(S)$ starts with a hole, then the result-table entry is $(0, 1)$, otherwise there is no entry.*
2. *Let T be a nonterminal with rule $T \to T_1T_2$.*
   - **a.** *All entries a, (a, b) in the prefix table of T_2 are inherited to the prefix table of T as $a + q$ and $(a + q, b)$, respectively, where $q = |val(T_1)|$.*
   - **b.** *An entry a of the prefix table of T_2 is tested by first adding a nonterminal A representing the prefix of $val(S)$ as suffix of $val(T_1)$, then $A' \to AT_2$ is added to the SLP. If $val(A')$ is a proper prefix of $val(S)$, then a is added to the prefix table of T. If $val(S)$ is a prefix of $val(A')$, then a is in the result table of T. Otherwise, no entry is inherited.*

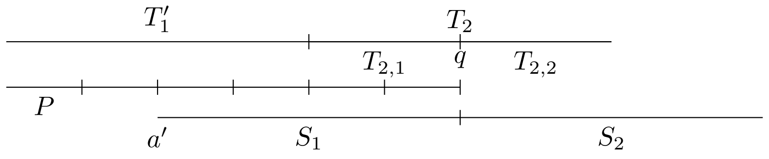
The complex case: *Let T be a nonterminal with rule $T \to T_1T_2$, and let there be an entry (a, b) in the list of the prefix table of T_1.*

Let T_1' be a nonterminal that represents the suffix of $val(T_1)$ starting at a. We determine the maximal number of periods corresponding to entry (a, b) that fit as prefix into $val(T_2)$. Therefore construct the potential period string as a nonterminal P as representing the string $val(T_1)[a, a + b - 1]$. Then construct nonterminals P_1 as a prefix and P_2 as a suffix of P, such that $val(P) = val(P_1P_2)$ and $val(T_1)[a..|val(T_1)| - 1] = val(P)^k val(P_1)$ for some k. Then construct P' as $P' \to P_2P_1$, and its powers on demand and use interval bisection to find the maximal prefix of $val(T_2)$ that is periodic with period string P'.



A similar computation has to be done for S: using interval bisection in order to find the maximal number k' of periods such that a prefix of $val(S)$ matches $val(P)^{k'}$. The possible case and corresponding outcomes and actions are:

1. *$val(T_2)$ is periodic with period string $val(P')$, and $val(S)$ has a proper prefix that matches the periodic string $val(T_1'T_2)$. The new entry for the prefix-table for T is (a, b). There will be no entry for the result-table.*
2. *$val(T_2)$ is periodic with period string $val(P')$, and $val(S)$ matches a prefix of $val(P)^k$, and item (1) does not hold. This means $val(S)$ is periodic as a partial word with period string $val(P)$.*
   Compute the number k by comparing the length of $val(S)$ with the length of $val(T_1'T_2)$ such that the arithmetic sequence $a, a+b, \ldots, a+(k-1)b$ represents the positions in $val(T_1'T_2)$ where S matches substrings of $val(T_1'T_2)$. The entry (a, b, k) is added to the result-table of T. The pair $(a+bk, b)$ is added as entry to the prefix-table of T.
3. *$val(T_2)$ is periodic with period string $val(P')$, and $val(S)$ has a proper prefix that matches $val(P)^m$ for some m, but not the the full periodic substring $val(T_1'T_2)$. Then compute the smallest k by simple arithmetic, such that $a+bk$ is position in $val(T_1'T_2)$, such that S matches the substring starting there. The entry $(a+bk, b)$ is added to the prefix-table of T. There is no result entry added.*
4. *$val(T_2)$ has a proper prefix that is periodic with string $val(P')$, but $val(T_2)$ is not periodic with period string $val(P')$, and $val(S)$ matches a prefix of $val(P)^k$ for some k.*
   This means $val(S)$ is periodic as a partial word with period string $val(P)$.
   Compute the number k using arithmetic, and interval bisection for $val(T_1'T_2)$, such that the arithmetic progression $a, a+b, \ldots, a+(k-1)b$ represents the positions in $val(T_1'T_2)$ where S matches substrings of $val(T_1'T_2)$. The entry (a, b, k) is added to the result-table of T. There are no entries in the prefix-table of T.
5. *$val(T_2)$ has a proper prefix that is periodic with string $val(P')$, $val(T_2)$ is not periodic with period string $val(P')$, and $val(S)$ does not match a prefix of $val(P)^k$ for any k.*



Then compute nonterminals S_1, S_2 extending the SLP, such that $val(S_1)$ is the maximal prefix of $val(S)$ that matches a prefix of $val(P)^m$ for some m, and $val(S) = val(S_1S_2)$. Also compute $T_{2,1}, T_{2,2}$ extending the SLP with $val(T_2) = val(T_{2,1})val(T_{2,2})$, and such that $val(T_{2,1})$ is the maximal prefix of $val(T_2)$ that is periodic with period string $val(P')$. Then there is at most one potential position for a matching occurrence (derived from the entries in T_1): $val(T_{2,2})$ and $val(S_2)$ must start at the same position. Let the position q be computed as $q := a + |val(T_1T_{2,1})|$. If $val(S_2)$ matches a prefix of $val(T_{2,2})$ or there is a prefix of $val(S_2)$ that matches $val(T_{2,2})$, and $val(S_1)$ matches a suffix of $val(T_1'T_{2,1})$, then we can proceed; otherwise, there is no entry. Let a' be the starting occurrence of $val(S)$ in $val(T_1'T_2)$, i.e. $a' := q - |val(S_1)|$.
If S matches $val(T_1T_2)$ at position a' and is completely contained, then a' is an entry in the result-table for T. If a proper prefix of S matches $val(T_1T_2)$ at position a' and is not completely contained, then a' will be an entry in the prefix-table of T.

Compaction *Finally, there is a compaction of the prefix table of T to keep the sum of the sizes of all prefix-tables polynomial. The compaction takes the prefix table as computed above as input and generates a completely new prefix table for T. Thereby it may generate also entries for arithmetic progressions in the prefix-table.*
Let h be the number of holes of $val(S)$. Let c be the current position in $val(T)$, where the

start is at position 0*, and where the compaction stops if* $|val(T)| - c < h + 3$ *and the input positions are simply moved to the output in this case.*
Let $d := |val(T)| - c + 1$ *and let* $e := \lfloor d/(2(h + 2)) \rfloor$*. If for* $val(T)$ *in the (position) interval* $c, c + e - 1$ *there are at least* $h + 3$ *positions for* S *including implicit positions from arithmetic progression, then the leftmost* $h + 2$ *entries in this interval are expanded, and* b' *is computed as the gcd of the position differences according to Theorem 3.10. If* p' *is the* $(h+3)^{th}$ *position, then the entry* (p', b') *is added to the output for* T*.*
Then do the same for all intervals $[c + ie, c + (i + 1)e - 1]$ *for* $i \in \{1, \dots, h + 1\}$*. This covers the left half of* $val(T)[d..|val(T)| - 1]$*. Continue with the same procedure for the right half of* $val(T)$*, but with refreshed* c, d, e*.*

The two tables are a complete description of all occurrences of prefixes of $val(S)$ as suffixes of $val(T)$ and of matching occurrences of S in T. The matching occurrences can be found in the result table of minimal nonterminals: If $T \to T_1 T_2$, then only the occurrences of S in T that cross the border between $val(T_1)$ and $val(T_2)$ are mentioned in the result-table of T. Other occurrences are mentioned at T_1, T_2 or their descendents.

▶ **Proposition 4.2.** *Let* S *be a nonterminal in* G_s *and* T *be a nonterminal in* G_t*, and assume the two tables are constructed with Algorithm 4.1. Then the prefix-table is a complete description of all occurrences of prefixes of* $val(S)$ *as suffixes of* $val(T)$*, and the result-tables of the nonterminals contributing to* T *(including* T*) are a complete description of all matching occurrences of* S *in* T*.*

Proof. The argument is by induction: Let $T \to T_1 T_2$ be a rule in G_t: We assume that the prefix-entries for T_1 and T_2 are complete, and then have to argue that the step of Algorithm 4.1 constructs a complete set of entries for T. This requires to verify that all the cases are treated. This is done in the algorithm due to the exact knowledge about periodicities. ◀

▶ **Proposition 4.3.** *The compaction step is correct. Also the semantics of the entries satisfied: for a periodic entry* (a, b)*, the suffix of* $val(T)$ *is periodic with period* b*. Moreover, the period is the optimal one, since the input contains all matching occurrences for prefixes of* $val(S)$*.*

Proof. This follows from Theorem 3.10. ◀

▶ **Proposition 4.4.** *The construction of the prefix- and result-table requires at most polynomial time in the size of* $G := G_s \cup G_t$*. For a fixed* S*, the size of the tables is of order* $O(|G|^4)$*.*

Proof. The operations can all be done in polynomial time, and the table has a polynomial number of places (see Lemma 2.7). What remains to be shown is that the number of entries in the list for nonterminal T is polynomial. Let n be the size of the SLP G and h be the number of holes of $val(S)$, and assume that the rule for detection of arithmetic sequences does not apply. Then we can determine an upper bound for the number of entries as follows: Here a single entry and a pair for arithmetic sequences count as one entry. Let $d = |val(T)|$, and let $e = d/(2(h + 2))$. In the interval $[0, e]$ at most $h + 3$ are starting, since otherwise a compaction step is possible. The same for all the intervals until $[(h + 1)e, (h + 2)e]$. Thus there are at most $(h + 2)(h + 3)$ entries in the left half of T. The same argument applies to the right half in the role of T. Due to the generation of strings by an SLP not more than n interval divisions by 2 are possible, hence at most $n * (h + 2)(h + 3)$ entries are in the table for the nonterminal T. ◀

4.1 Results for Matching of Strings with Holes

▶ **Theorem 4.5.** *The LCCSM problem can be solved in polynomial time: If s is a string with n holes, and t is a string, compressed by SLPs G_s and G_t, respectively, then a representation of all the matching occurrences of s in t can be computed in polynomial time.*

Proof. Using Algorithm 4.1 for the construction of the result-table is complete and requires polynomial time in $|G_t| + |G_s|$ by Proposition 4.4. ◀

Note that the number of matching occurrences may be exponential in $|G_t| + |G_s|$. The computed table represents all occurrences in a compact way in polynomial space.

A representation of all periods of a compressed partial word s can be computed: Let S be a nonterminal with $val(S) = s$. Compute nonterminals $S_i, i = 1, \ldots, m$ with $m \leq |G|$ that represent the maximal hole-free substrings of s. Then compute all the periods of $val(S_i)$ for $i = 1$ using a fully compressed matching algorithm [17], also Algorithm 4.1 could be specialized to this. This results in a polynomial number of potential periods. Now test every potential period p as follows: First compute a nonterminal S' representing S where the holes are instantiated by characters using the period p. Then compute a nonterminal P with $val(P) = val(S)[0..p-1]$ and check whether $val(P^k P') =_h val(S)$ where P' represents a prefix of $val(P)$ and $|val(P^k P')| = |val(S)|$. For every period of $val(S)$, at least one divisor will be found, their number is at most polynomial and every check can be done in PTIME. Note that Algorithm 4.1 cannot be used to find periodicities of S in S (see item 5.3).

▶ **Proposition 4.6.** *All periods of an SLP-compressed string s can be computed in PTIME.*

5 Extensions and Generalizations

5.1 Adding Variables Representing Strings

▶ **Lemma 5.1.** *Let s be a string linear in the variables, every variable is a string-variable, let t be a string, and assume that both are compressed by an SLP. Then the question whether there exists a substitution σ, such that $\sigma(s)$ occurs in t can be decided in polynomial time.*

Proof. Let S, T be nonterminals with $val(S) = s$, $val(T) = t$. An eager left-to-right search is sufficient: If $val(S) = s_1 x_1 \ldots x_n s_{n+1}$, where s_i are ground, then first construct nonterminals S_i with $val(S_i) = s_i$ for $i = 1, \ldots, n$. Then identify the first occurrence of s_1 in t. The next step is to construct a nonterminal representing the suffix of t right to the occurrence and repeat the search. This is a sequential algorithm, every step can be performed in polynomial time, and the required SLP does not grow in size. Hence whether a match exists can be decided in polynomial time, and a match can be computed as a side-effect. ◀

This can be generalized to a mix of string-variables and character-variables:

▶ **Proposition 5.2.** *Let s be a string linear in the variables, which may be string-variables or character-variables, let t be a string, and assume s, t are SLP-compressed. Then the question whether there exists a substitution σ, such that $\sigma(s)$ occurs in t can be decided in PTIME.*

Proof. Similar to the previous proof: let $s = s_1 x_1 \ldots x_n s_{n+1}$, where x_i are all the string-variables, and s_i may contain character-variables. Then an eager left-to-right search will run in polynomial time, using Theorem 4.5, similar as in the proof of Lemma 5.1. ◀

5.2 Nonlinear Patterns

Suppose, we permit multiple occurrences of the same character-variable in the pattern. Note that the number of different variables is at most $|G_s|$, however, there may be an exponential number of occurrences of variables in s. Unfortunately, Theorem 3.10 is no longer applicable. If the number of occurrences is polynomial, then we can use our current method as follows:

▶ **Proposition 5.3.** *Let the number of occurrences of variables in s be k. Then the pattern match problem can be decided in time $O(poly_k(|G_s|, |G_t|))$ where poly is a polynomial.*

Proof. Linearize the pattern s giving s', such that $\rho(s') = s$ for some variable-variable substitution. Perform a pattern match using s', resulting in a polynomially-sized result-table. Then scan every entry: For single entries a, test whether s' occurs in t at position a, which is polynomial. For arithmetic sequences of length $< k + 2$ use the method for single entries, otherwise a unique substitution can be computed from the sequence. Then test whether this substitution is compatible with ρ. All the tests can be done in polynomial time. ◀

5.3 Further Possible Extension and Problems

Pattern Target is a Partial Word. If the target t of the pattern match is a partial word, then our method does no longer work as expected, since the overlap-theorem can no longer by applied. For example let $s = aa\circ bb$, and $t = aaa\circ\circ bb$, then s matches t at positions $0, 1, 2$. However, the three occurrences of s do not overlap, and so the overlap theorem cannot be used for compacting the entries in the prefix-table in Algorithm 4.1.

Compressed Partial Words Including Holes If holes are not seen as variables, but represented as an extra symbol, then compression can represent partial words with a larger number of holes, up to an exponential number of holes. However, then our arguments fail, in particular the application of the overlap-theorem 3.10 no longer helps in compacting the prefix-table entries, since the number of holes is no longer $O(|G_s|)$.

Term Matching Looking for submatching (or the encompassment relation) of terms as a generalisation of strings is of course an interesting generalization. The technical report [24] contains an approach to compressed term matching but has to leave open the issue whether a single compressed term submatch can be performed in polynomial time and even the question for terms with linear variable occurrence was left open.

6 Conclusion and Further Work

Future work is to analyze the CCSM problem for strings. Another important issue is to analyze term-submatching, to generalize CCSM and LCCSM to terms, and then to improve the algorithms in [24].

Acknowledgements I thank David Sabel for discussions on compressions and for reading the paper.

References

1 D. Benanav, D. Kapur, and P. Narendran. Complexity of matching problems. *J. Symb. Comput.*, 3(1/2):203–216, 1987.

2 P. Berman, M. Karpinski, L. L. Larmore, W. Plandowski, and W. Rytter. On the complexity of pattern matching for highly compressed two-dimensional texts. *J. Comput. Syst. Sci.*, 65(2):332–350, 2002.

3 J. Berstel and L. Boasson. Partial words and a theorem of Fine and Wilf. *Theor. Comput. Sci.*, 218(1):135–141, 1999.

4 Ph. Bille, R. Fagerberg, and I. Li Gørtz. Improved approximate string matching and regular expression matching on Ziv-Lempel compressed texts. *ACM Transactions on Algorithms*, 6(1), 2009.

5 F. Blanchet-Sadri. Periodicity on partial words. *Comput. Math. Appl.*, 47:71–82, 2004.

6 F. Blanchet-Sadri. *Algorithmic combinatorics on partial words.* Chapman & Hall/CRC, 2008.

7 F. Blanchet-Sadri, D. Dakota Blair, and R. V. Lewis. Equations on partial words. In *MFCS*, *LNCS* 4162 , pages 167–178. Springer, 2006.

8 F. Blanchet-Sadri and R. A. Hegstrom. Partial words and a theorem of Fine and Wilf revisited. *Theor. Comput. Sci.*, 270(1-2):401–419, 2002.

9 G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML documents. In *Proceedings of DBPL 2005*, *LNCS* 3774, pages 199–216, 2005.

10 N. J. Fine and H. S. Wilf. Uniqueness theorem for periodic functions. *Proc. Am. Math. Soc.*, 16:109–114, 1965.

11 A. Gascón, G. Godoy, and M. Schmidt-Schauß. Unification and matching on compressed terms. *ACM Trans. Comput. Log.*, 12(4):26:1–26:37, 2011.

12 A. Jez. Faster fully compressed pattern matching by recompression. *CoRR*, abs/1111.3244, 2011.

13 M. Karpinski, W. Rytter, and A. Shinohara. Pattern-matching for strings with short description. In *CPM '95*, *LNCS* 937, pages 205–214. Springer-Verlag, 1995.

14 M. Karpinski, W. Rytter, and A. Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nord. J. Comput.*, 4(2):172–186, 1997.

15 J. Levy, M. Schmidt-Schauß, and M. Villaret. Monadic second-order unification is NP-complete. In *RTA-15*, *LNCS* 3091, pages 55–69. Springer, 2004.

16 J. Levy, M. Schmidt-Schauß, and M. Villaret. The complexity of monadic second-order unification. *SIAM J. of Computing*, 38(3):1113–1140, 2008.

17 Y. Lifshits. Processing compressed texts: A tractability border. In *CPM 2007*, pages 228–240, 2007.

18 M. Lohrey. Word problems and membership problems on compressed words. *SIAM Journal on Computing*, 35(5):1210–1240, 2006.

19 M. Lohrey. Compressed membership problems for regular expressions and hierarchical automata. *Int. J. Found. Comput. Sci.*, 21(5):817–841, 2010.

20 N. Markey and Ph. Schnoebelen. A PTIME-complete matching problem for SLP-compressed words. *Inf. Process. Lett.*, 90(1):3–6, 2004.

21 G. Navarro. Regular expression searching on compressed text. *J. Discrete Algorithms*, 1(5-6):423–443, 2003.

22 W. Plandowski. Testing equivalence of morphisms in context-free languages. In *ESA 94*, volume 855 of *Lecture Notes in Computer Science*, pages 460–470, 1994.

23 W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. In *Jewels are Forever*, pages 262–272. Springer, 1999.

24 M. Schmidt-Schauß. Pattern matching of compressed terms and contexts and polynomial rewriting. Frank report 43, Institut für Informatik. Goethe-Univ. Frankfurt am Main, 2011.

25 T. Yamamoto, H. Bannai, S. Inenaga, and M. Takeda. Faster subsequence and don't-care pattern matching on compressed texts. In *22nd CPM*, *LNCS* 6661, pages 309–322. Springer, 2011.

26 J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

27 J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

Meaningless Sets in Infinitary Combinatory Logic

Paula Severi[1,2] and Fer-Jan de Vries[1,3]

1 **Department of Computer Science, University of Leicester, UK**
2 `ps56@mcs.le.ac.uk`
3 `fdv1@mcs.le.ac.uk`

Abstract

In this paper we study meaningless sets in infinitary combinatory logic. So far only a handful of meaningless sets were known. We show that there are uncountably many meaningless sets. As an application to the semantics of finite combinatory logics, we show that there exist uncountably many combinatory algebras that are not a lambda algebra. We also study ways of weakening the axioms of meaningless sets to get, not only sufficient, but also necessary conditions for having confluence and normalisation.

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.1.1 Models of Computation, I.2.2 Automatic Programming, I.2.3 Deduction and Theorem Proving, I.2.4 Knowledge Representation Formalisms and Methods

Keywords and phrases infinitary rewriting, infinitary combinatory logic, meaningless set confluence and normalisation

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.288

Category Regular Research Paper

1 Introduction

In this paper, we study *meaningless sets* for *infinitary combinatory logic* [8, 5]. This is of interest because for infinitary combinatory logic, only a handful of meaningless sets are known so far, in stark contrast to the current situation for infinitary lambda calculus [14, 16]. Meaningless sets play an important role in the construction of *syntactic models* of finite lambda calculus and combinatory logic. The *interpretation* of a term is its infinitary normal form and it is well-defined if the corresponding infinitary lambda calculus or infinitary rewriting system is *confluent* and *normalising*. The extension with infinite terms and infinite reductions ruins the confluence property. We can recover confluence by extending the syntax with a fresh symbol $\perp$ and by extending the reduction rules with a $\perp$-rule that can reduce terms from a well chosen set $\mathcal{U}$ of meaningless terms to $\perp$. The papers [8, 5] looked simultaneously at orthogonal infinitary term rewriting systems and infinitary lambda calculus and gave a sufficient set of conditions for $\mathcal{U}$ so that the corresponding infinitary term rewriting systems or infinitary lambda calculi are confluent and normalising. Later for infinitary lambda calculus it was found in [16] that there exists a set of *necessary and sufficient conditions* for $\mathcal{U}$ such that the corresponding infinitary lambda calculus with β and $\perp$-reduction is confluent and normalising. And in [14, 15] it has been shown that there are uncountably many of such sets and hence *uncountably* many syntactic models of the lambda calculus. These sufficient and necessary conditions in the case of lambda calculus do not immediately carry over to orthogonal term rewriting systems. This hinges on the fact that in infinitary lambda calculus the set of rootactive terms coincides with the set of hypercollapsing terms. This is in general not the case for arbitrary orthogonal infinitary rewriting systems: infinitary combinatory logic is an example of an infinitary rewriting system with rootactive terms that are not hypercollapsing. The reduction for combinatory logic is called w-reduction [1, 9]. In this paper we look at the following questions: what are the (necessary and) sufficient conditions for the infinitary combinatory logic with $w\bot$-reduction to be (1) confluent?, (2) normalising? and (3) the combination, confluent and normalising?.

Regarding confluence we will show that infinitary combinatory logic with $w\bot$-reduction is confluent for the traditional set of axioms of meaningless sets without assuming the axiom of rootactiveness. We give *uncountably* many examples of meaningless sets satisfying rootactiveness. Each one gives rise to a model of finite combinatory logic, called *combinatory algebra* [1]. We also show that these models are not λ-algebras, i.e. there exist two terms whose corresponding translations into lambda calculus are β-convertible but they have different interpretation in the model.

By studying the overlap cases between w and $\bot$-reduction, we realise that we can weaken the axiom of overlap. Combining weak overlap with rootactiveness and the axioms of closure under reduction, expansion and indiscernibility gives a *sufficient condition* so that the infinitary combinatory logic with $w\bot$-reduction is confluent and normalising. We also show that the axioms of hypercollapseness, overlap and closure under reduction are *necessary conditions* for confluence.

The paper is organised as follows. Section 2 gives a brief overview of infinitary rewriting in the setting of combinatory logic. Section 3 works out what the traditional theory of meaningless sets means for infinitary combinatory logic. Section 4 studies the axioms of closure under expansion and substitution. Section 5 gives concrete examples of meaningless sets and describes some of the structure of the lattice of meaningless sets. Section 6 shows an application to combinatory algebras. Section 7 explores sufficient and necessary conditions for confluence. Section 8 discusses some open problems and shows an example of a normalising infinitary combinatory logic that does not satisfy rootactiveness.

2 Infinitary Combinatory Logic

We will define infinitary combinatory logic assuming familiarity with basic notions and notations from combinatory logic, lambda calculus [1, 9] and some familiarity of infinite term rewriting [5]. The set $\mathcal{CL}$ of finite CL-terms is defined by induction from the following grammar:

$$M ::= x \mid \mathbf{K} \mid \mathbf{S} \mid MM.$$

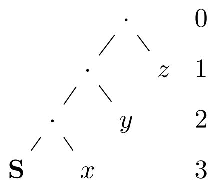
The set $\mathcal{CL}^\infty$ of finite and infinite CL-terms over a given set of variables is defined by coinduction from the same grammar.

▶ **Definition 2.1** (w-reduction)**.** The w-rules are extended to $\mathcal{CL}^\infty$.

$$\begin{aligned}&\mathbf{K}MN \to_w M\\&\mathbf{S}MNP \to_w MP(NP)\end{aligned}$$

The w in $\to_w$ stands for weak reduction [1, 3]. The first rule is called $\mathbf{K}$-*rule* and the second $\mathbf{S}$-*rule*. A term is a $\mathbf{K}$-redex (or $\mathbf{S}$-redex) if it is of the form $\mathbf{K}MN$ (or $\mathbf{S}MNP$).

Above we have presented $\mathcal{CL}$ in the traditional applicative format: this means that the infix application symbol $\cdot$ is suppressed, outermost brackets dropped and the usual bracket convention of association to the left is followed: e.g., xyz actually stands for $((x \cdot y) \cdot z)$. Alternatively we could have presented the terms of $\mathcal{CL}$ and $\mathcal{CL}^\infty$ and the two rules in the format of first order term rewriting by adding an explicit binary application symbol $\mathbf{Ap}$ to the syntax $M ::= x \mid \mathbf{K} \mid \mathbf{S} \mid \mathbf{Ap}(M, N)$ with the proviso that we will read MN for $\mathbf{Ap}(M, N)$ [9]. Terminology and notation for infinite term rewriting in the latter format translate readily to CL presented in its applicative format. E.g. we say that $\mathbf{S}$ and x have depth 3 in the term $\mathbf{S}xyz$, whereas the depth of y and z is respectively 1 and 2.



The distance $d(M, N)$ between two terms is defined as 2^{-n} where n is the length of the shortest common position p of M and N such that M and N differ at p. With this notion of distance the set $\mathcal{C}L$ of finite terms becomes a metric space and $\mathcal{C}L^\infty$ its metric completion.

Metavariables in (infinitary) combinatory logic will be denoted by $M, N, P, \ldots$ and contexts by $C, D, \ldots$. Simultaneous substitution will be denoted by M^σ, where σ is a substitution of variables by terms.

One step reduction $\to_w$ on $\mathcal{C}L$ and $\mathcal{C}L^\infty$ is the smallest binary relation on $\mathcal{C}L$, respectively $\mathcal{C}L^\infty$ containing the w-rules and closed under contexts. Let α be an ordinal.

▶ **Definition 2.2** (Strongly converging reduction sequence). A *strongly converging reduction sequence* of length α is a sequence of reduction steps $(M_\beta \to_w M_{\beta+1})_{\beta<\alpha}$ such that for every limit ordinal $\lambda \leq \alpha$:

1. the sequence of terms $(M_\beta)_{\beta\leq\lambda}$ is a transfinite Cauchy sequence, that is
$$\lim_{\beta\to\lambda} M_\beta = M_\lambda$$
2. $\lim_{\beta\to\lambda} d_\beta = \infty$ where d_β is the depth of the redex contracted at $M_\beta \to_w M_{\beta+1}$.

We will denote this by $M_0 \twoheadrightarrow\!\!\!\!\rightarrow_w^\alpha M_\alpha$ (or just $M_0 \twoheadrightarrow\!\!\!\!\rightarrow_w M_\alpha$). We will use the notation $M \twoheadrightarrow_w N$ for a finite reduction from M to N.

Because $\mathcal{C}L$ is a left-linear system the Compression Lemma holds, which says that whenever $M_0 \twoheadrightarrow\!\!\!\!\rightarrow_w^\alpha M_\alpha$ then $M_0 \twoheadrightarrow\!\!\!\!\rightarrow_w^{\leq\omega} M_\alpha$, i.e. there is an strongly converging reduction from M_0 to M_α of length at most ω.

We will use the following abbreviations for terms:

$$\begin{array}{ll} \mathbf{I} = \mathbf{SKK}, & \mathbf{\Omega} = \mathbf{SII}(\mathbf{SII}), \\ M^\omega = M(M(M(\ldots))) & C^\omega = C[C[C[\ldots]]] \text{ if } C[\ ] \text{ is a context.} \end{array}$$

Although the names $\mathbf{I}$ and $\mathbf{\Omega}$ may feel familiar from lambda calculus, one should beware that their reduction behaviour is slightly different: $\mathbf{I}x \twoheadrightarrow_w x$ takes two steps, while $\mathbf{\Omega} \twoheadrightarrow_w \mathbf{\Omega}$ takes five. For the complete reduction graph of $\mathbf{\Omega}$ in $\mathcal{C}L$ see, see [9]. We give some examples of interesting infinite terms.

▶ **Example 2.3.** 1. It is easy to verify that the infinite term $\mathbf{Y} = (\mathbf{SI})^\omega$ is an infinite fixed point combinator

$$(\mathbf{SI})^\omega M = \mathbf{SI}(\mathbf{SI})^\omega M \to_w \mathbf{I}M((\mathbf{SI})^\omega M) \to_w M((\mathbf{SI})^\omega M).$$

To verify that a finite term is a fixed point combinators usually takes more steps to prove. See for instance the shortest fixed point combinator [17].

$$\mathbf{Y}_{Tromp} = \mathbf{SSK}(\mathbf{S}(\mathbf{K}(\mathbf{SS}(\mathbf{S}(\mathbf{SSK}))))\mathbf{K}).$$

The infinite term $(\mathbf{SI})^\omega$ can be obtained as limit of a strongly convergent reduction starting from the finite term $\mathbf{Y}_{Tromp}(\mathbf{SI})$.

2. Since $\mathbf{Y}(\mathbf{SII}) \twoheadrightarrow_w (\mathbf{Y}(\mathbf{SII}))(\mathbf{Y}(\mathbf{SII}))$ we find that if we continue this process, $\mathbf{Y}(\mathbf{SII})$ strongly converges to an infinite normal form, the binary tree of applications:
$$\mathbf{X} = (((\ldots)(\ldots))((\ldots)(\ldots)))(((\ldots)(\ldots))((\ldots)(\ldots))).$$
3. The term $\mathbf{S}^\omega\mathbf{XX}$ is self-looping: $\mathbf{S}^\omega\mathbf{XX} = \mathbf{S}(\mathbf{S}^\omega)\mathbf{XX} \to_w (\mathbf{S}^\omega)\mathbf{X}(\mathbf{XX}) = \mathbf{S}^\omega\mathbf{XX}$.
4. Let $C_M[\ ]$ abbreviate the context $\mathbf{K}[\ ]M$. Then C_M^ω is self-looping for any M. Redexes of the form $C_M[x] \to_w x$ are called *collapsing*.

Let Λ^∞ is defined by coinduction from the following grammar.

$M ::= \perp \mid x \mid (\lambda x M) \mid (MM)$

Combinator terms translate directly into lambda terms.

▶ **Definition 2.4** (Translation from $\mathcal{C}L^\infty$ to Λ^∞). For $M \in \mathcal{C}L^\infty$ we define $M_\lambda \in \Lambda^\infty$ by coinduction as follows: $x_\lambda = x$, $\mathbf{K}_\lambda = \lambda xy.x$, $\mathbf{S}_\lambda = \lambda xyz.xz(yz)$, $(PQ)_\lambda = P_\lambda Q_\lambda$.

We define $\to_w^h$ as the restriction of $\to_w$ to w-redexes in the head position where M is in the head position of N if $N = MP_1 \ldots P_n$.

▶ **Definition 2.5** (Skeleton). We define the skeleton of a term by corecursion as follows.

$\mathsf{skel}(M) =$	N	if $M \twoheadrightarrow_w^h N$, and N is either x, $\mathbf{K}$ or $\mathbf{S}$
$\mathsf{skel}(M) =$	$\mathsf{skel}(N)\mathsf{skel}(P)$	if $M \twoheadrightarrow_w^h NP$, while $NP \not\twoheadrightarrow_w^h Q$ for any w-redex Q
$\mathsf{skel}(M) =$	M	otherwise

The skeleton of a term can be computed by a depth-first leftmost strategy that contracts all w-redexes not contained in a rootactive term (see Definition 3.4). We have that $M \twoheadrightarrow_w \mathsf{skel}(M)$.

3 Axioms of Meaningless Sets in Infinitary Combinatory Logic

Finite combinatory logic $\mathcal{C}L$ is confluent for finite w-reduction. In contrast infinitary combinatory logic $\mathcal{C}L^\infty$ is not confluent for strongly converging w-reduction. The reason behind this negative result is that the $\mathbf{K}$-rule is a collapsing rule with a multivariable left-hand side. The infinite tower of collapsing redexes $C_x[C_y[C_x[C_y[\ldots]]]]$, that is the term $\mathbf{K}(\mathbf{K}(\mathbf{K}(\mathbf{K}(\ldots)y)x)y)x$, can strongly converge to both C_x^ω and C_y^ω, which are self-looping by Example 2.3. Hence C_x^ω and C_y^ω can not reduce further to a common reduct [6].

▶ **Definition 3.1** (Hypercollapsing Term). We say that a term $M \in \mathcal{C}L_\perp^\infty$ is hypercollapsing, if any reduct of M can further reduce to a collapsing redex. We denote the set of hypercollapsing terms by $\mathcal{H}$.

Confluence can be restored if we identify hypercollapsing terms with $\perp$ [8]. For this, we extend $\mathcal{C}L^\infty$ with a fresh symbol $\perp$. We define $\mathcal{C}L_\perp^\infty$ by coinduction from the grammar: $M ::= x \mid \perp \mid \mathbf{K} \mid \mathbf{S} \mid MN$. We also add a reduction rule $\to_{\perp_\mathcal{U}}$ that allows us reduce terms from a chosen subset $\mathcal{U} \subseteq \mathcal{C}L^\infty$ to $\perp$. This is reminiscent to $B\Omega$-reduction in [1] where the unsolvables play the role of undefined terms. We need some notation to define this $\perp$-rule.

▶ **Definition 3.2** ($\perp$-instance)**.** Let $\mathcal{U} \subseteq \mathcal{CL}^{\infty}$ and $M, N \in \mathcal{CL}^{\infty}_{\perp}$. We say that M is a $\perp$-instance of N, notation $M \preceq_{\mathcal{U}} N$, if M is obtained from N by replacing some (possibly infinitely many) subterms of N in $\mathcal{U}$ by $\perp$. We define $\mathcal{U}_{\perp} \subseteq \mathcal{CL}^{\infty}_{\perp}$ as $\mathcal{U}_{\perp} = \{M \mid \exists N \in \mathcal{U}.M \preceq_{\mathcal{U}} N\}$.

▶ **Definition 3.3** ($\perp_{\mathcal{U}}$-reduction)**.** Let $\mathcal{U} \subseteq \mathcal{CL}^{\infty}$. We define the $\perp_{\mathcal{U}}$-rule on $\mathcal{CL}^{\infty}_{\perp}$: $M \to \perp$ if $M \in \mathcal{U}_{\perp}$ and $M \neq \perp$. The one step reduction $\to_{\perp_{\mathcal{U}}}$ is the smallest binary relation containing $\perp_{\mathcal{U}}$ and closed under contexts and substitutions.

Occasionally, we may denote $\perp_{\mathcal{U}}$ just by $\perp$. We denote finite $\perp$-reductions by $\twoheadrightarrow_{\perp}$ and strongly convergent reductions by $\twoheadrightarrow\!\!\!\!\rightarrow_{\perp}$.

▶ **Definition 3.4** (Rootactive Term)**.** We say that a term $M \in \mathcal{CL}^{\infty}_{\perp}$ is rootactive, if any reduct of M can further reduce to a redex, i.e. either a $\mathbf{K}$ or an $\mathbf{S}$-redex. We denote the set of rootactive terms by $\mathcal{R}$.

All hypercollapsing terms are rootactive. The converse is not true. Examples of terms that are rootactive but not hypercollapsing are the term $\mathbf{\Omega}$, any term of the form $\mathbf{S}^{\omega}MN$ (in particular, the term $\mathbf{S}^{\omega}\mathbf{X}\mathbf{X}$) and terms of the form $\mathbf{D}^{\omega}M$ (where $\mathbf{D}^{\omega}$ is the infinite tower of contexts of the form $\mathbf{S}(\mathbf{K}[\ \ ])\mathbf{I})$) for any M.

Note that unlike in lambda calculus [7, 8] we have that in combinatory logic the set of hypercollapsing terms does not coincide with the rootactive terms..

▶ **Definition 3.5** (Hypercollapseness)**.** We say that a set $\mathcal{U} \subseteq \mathcal{CL}^{\infty}$ satisfies the Axiom of hypercollapseness if $\mathcal{H} \subseteq \mathcal{U}$.

▶ **Definition 3.6** (Rootactiveness)**.** We say that a set $\mathcal{U} \subseteq \mathcal{CL}^{\infty}$ satisfies the Axiom of rootactiveness if $\mathcal{R} \subseteq \mathcal{U}$.

▶ **Definition 3.7** ($\overset{\mathcal{U}}{\longleftrightarrow}$ and $\overset{\mathcal{U}}{=}$)**.** Let $M, N \in \mathcal{CL}^{\infty}_{\perp}$. We write $M \overset{\mathcal{U}}{\longleftrightarrow} N$, if N can be obtained from M by replacing some (possibly infinitely many) subterms of M in $\mathcal{U}$ by other terms in $\mathcal{U}$. We write $t \overset{\mathcal{U}}{=} s$ for the transitive closure of $\overset{\mathcal{U}}{\longleftrightarrow}$.

▶ **Definition 3.8** (Indiscernibility)**.** We say that a set $\mathcal{U} \subseteq \mathcal{CL}^{\infty}$ satisfies the Axiom of indiscernibility if for all $M, N \in \mathcal{CL}^{\infty}$ such that $M \overset{\mathcal{U}}{\longleftrightarrow} N$, we have that $M \in \mathcal{U}$ iff $N \in \mathcal{U}$.

▶ **Definition 3.9** (Closure under w-reduction)**.** We say that a set $\mathcal{U} \subseteq \mathcal{CL}^{\infty}$ satisfies the Axiom of closure under w-reduction if $M \in \mathcal{U}$ and $M \twoheadrightarrow\!\!\!\!\rightarrow_{w} N$ implies $N \in \mathcal{U}$ for all $M, N \in \mathcal{CL}^{\infty}$.

The general formulation of the next axiom of overlap says that if a redex M overlaps a subterm, and this subterm is in $\mathcal{U}$ then $M \in \mathcal{U}$. For combinatory logic this means concretely:

▶ **Definition 3.10** (Overlap)**.** We say that a set $\mathcal{U} \subseteq \mathcal{CL}^{\infty}$ satisfies the axiom of overlap if the following conditions holds for all $M, N, P \in \mathcal{CL}^{\infty}$:
1. If $\mathbf{K} \in \mathcal{U}$ or $\mathbf{K}M \in \mathcal{U}$ then $\mathbf{K}MN \in \mathcal{U}$.
2. If $\mathbf{S} \in \mathcal{U}$, $\mathbf{S}M \in \mathcal{U}$ or $\mathbf{S}MN \in \mathcal{U}$ then $\mathbf{S}MNP \in \mathcal{U}$.

▶ **Definition 3.11** (Meaningless Set)**.** A set $\mathcal{U} \subseteq \mathcal{CL}^{\infty}$ is called a *set of meaningless terms* (meaningless set for short), if it satisfies the axioms of hypercollapseness, closure under w-reduction, indiscernibility and overlap. These four axioms are called *the axioms of meaninglessness.*

▶ **Theorem 3.12** (Meaninglessness implies Confluence modulo $\mathcal{U}$ [8, 5])**.** *If $\mathcal{U}$ is a set of meaningless terms, then $(\mathcal{CL}^{\infty}_{\perp}, \twoheadrightarrow\!\!\!\!\rightarrow_{w})$ is confluent modulo $\mathcal{U}$, i.e. if $M \leftarrow\!\!\!\!\twoheadleftarrow_{w} \overset{\mathcal{U}}{=} \twoheadrightarrow\!\!\!\!\rightarrow_{w} N$ implies $P \twoheadrightarrow\!\!\!\!\rightarrow_{w} \overset{\mathcal{U}}{=} \leftarrow\!\!\!\!\twoheadleftarrow_{w} Q$.*

▶ **Theorem 3.13** (Indiscernibility implies Confluence of $\perp_{\mathcal{U}}$-reduction [8, 5]). *Let $\mathcal{U} \subseteq \mathcal{C}L^{\infty}$. If $\mathcal{U}$ satisfies indiscernibility then $(\mathcal{C}L^{\infty}_{\perp}, \twoheadrightarrow_{\perp_{\mathcal{U}}})$ is confluent.*

▶ **Theorem 3.14** (Postponement [8, 5]). *Let $\mathcal{U} \subseteq \mathcal{C}L^{\infty}$. If $M \twoheadrightarrow_{w\perp_{\mathcal{U}}} N$ then there exists P such that $M \twoheadrightarrow_{w} P \twoheadrightarrow_{\perp_{\mathcal{U}}} N$. (No properties of $\mathcal{U}$ need to be assumed.)*

▶ **Theorem 3.15** (Rootactiveness implies Normalization [8, 5]). *If $\mathcal{U} \subseteq \mathcal{C}L^{\infty}$ satisfies rootactiveness, then $(\mathcal{C}L^{\infty}_{\perp}, \twoheadrightarrow_{w\perp_{\mathcal{U}}})$ is normalising.*

▶ **Theorem 3.16** (Rootactiveness and Meaninglessness implies Confluence [8, 5]). *If $\mathcal{U} \subseteq \mathcal{C}L^{\infty}$ is a meaningless set that satisfies rootactiveness, then $(\mathcal{C}L^{\infty}_{\perp}, \twoheadrightarrow_{w\perp_{\mathcal{U}}})$ is confluent.*

▶ Notation 3.17 (Normal Form). If $(\mathcal{C}L^{\infty}_{\perp}, \twoheadrightarrow_{w\perp_{\mathcal{U}}})$ is confluent and normalising, then every term M in $\mathcal{C}L^{\infty}_{\perp}$ has a unique normal form, that we denote by $\mathsf{nf}_{\mathcal{U}}(M)$. We also write $\mathsf{nf}_{\mathcal{U}}(X) = \{\mathsf{nf}_{\mathcal{U}}(M) \mid M \in X\}$ for $X \subseteq \mathcal{C}L^{\infty}_{\perp}$.

Next we prove that $(\mathcal{C}L^{\infty}_{\perp}, \twoheadrightarrow_{w\perp_{\mathcal{U}}})$ is confluent for meaningless sets $\mathcal{U}$ that do not necessarily satisfy rootactiveness with tools from from [8, 5]. The earlier papers bypassed this result, because their interest lay in extensions $(\mathcal{C}L^{\infty}_{\perp}, \twoheadrightarrow_{w\perp_{\mathcal{U}}})$ that are both confluence and normalising.

▶ **Theorem 3.18** (Meaninglessness implies Confluence). *If $\mathcal{U}$ is a meaningless set, then $(\mathcal{C}L^{\infty}_{\perp}, \twoheadrightarrow_{w\perp_{\mathcal{U}}})$ is confluent.*

Proof. Let $M \xrightarrow{out_{\mathcal{U}}}_{w} N$ denote one w-step where the contracted redex in $M \to_{w} N$ is not contained in any subterm P of M which is in $\mathcal{U}$.

$$\begin{tikzcd}
 & & & M \arrow[dlll, two heads, "w\perp_{\mathcal{U}}"'] \arrow[dl, dotted, two heads, "w"'] \arrow[dr, dotted, two heads, "w"] \arrow[drrr, two heads, "w\perp_{\mathcal{U}}"] & & & \\
\bullet \arrow[d, dotted, two heads, "w"'] \arrow[rr, phantom, "(1)"] & & \bullet \arrow[ll, dotted, two heads, "\perp_{\mathcal{U}}"'] \arrow[d, dotted, two heads, "w"', "out_{\mathcal{U}}"] \arrow[rr, phantom, "(2)"] & & \bullet \arrow[rr, dotted, two heads, "\perp_{\mathcal{U}}"] \arrow[d, dotted, two heads, "out_{\mathcal{U}}"', "w"] & & \bullet \arrow[d, dotted, two heads, "w"] \arrow[ll, phantom, "(1)"'] \\
\bullet \arrow[dr, dotted, two heads, "\perp_{\mathcal{U}}"'] \arrow[rr, phantom, "(3)"] & & \bullet \arrow[ll, dotted, two heads, "\perp_{\mathcal{U}}"'] \arrow[dr, dotted, two heads, "\perp_{\mathcal{U}}"] & & \bullet \arrow[rr, dotted, two heads, "\perp_{\mathcal{U}}"] \arrow[dl, dotted, two heads, "\perp_{\mathcal{U}}"'] & & \bullet \arrow[dl, dotted, two heads, "\perp_{\mathcal{U}}"] \arrow[ll, phantom, "(3)"'] \\
 & \bullet \arrow[drr, dotted, two heads, "\perp_{\mathcal{U}}"'] \arrow[r, phantom, "(4)"] & & \bullet \arrow[ll, dotted, two heads, "\perp_{\mathcal{U}}"'] \arrow[rr, dotted, two heads, "\perp_{\mathcal{U}}"] \arrow[d, phantom, "(4)"] & & \bullet \arrow[dll, dotted, two heads, "\perp_{\mathcal{U}}"] \arrow[l, phantom, "(4)"'] & \\
 & & & \bullet & & &
\end{tikzcd}$$

Suppose we are given two $w\perp_{\mathcal{U}}$-reductions starting from M. Using Postponement Theorem 3.14 we factor both reductions in a w-reduction followed by a $\perp_{\mathcal{U}}$ reduction. This gives us the two triangles (1). Next, from Theorem 3.12, we find two w-reductions ending in terms that are identical up to subterms in $\mathcal{U}$. If we factor both reductions into an outside $\mathcal{U}$ w-reduction (Lemma 12.9.20 in [5], which needs overlap and closure under reduction), followed by an inside $\mathcal{U}$ w-reduction we obtain (2). And (3) follows from commutation of outside $\mathcal{U}$ w-reduction with $\twoheadrightarrow_{\perp_{\mathcal{U}}}$ (Lemma 12.9.19 in [5], which needs overlap), while (4) is implied by Theorem 3.13 (confluence of $\twoheadrightarrow_{\perp}$). ◀

Note that the previous theorem and proof holds for arbitrary orthogonal term rewriting systems as well.

▶ Remark. In [16], we could prove that confluence implies normalisation for infinitary lambda calculus with $\beta\bot_{\mathcal{U}}$-reduction defined with any set of finite and infinite terms $U \subseteq \Lambda^\infty$. In the case of combinatory logic, this does not hold. For example, take $\mathcal{U} = \mathcal{H}$. By Theorem 3.18 we have that $(\mathcal{C}L^\infty_\bot, \twoheadrightarrow\!\!\!\!\!\!\rightarrow_{w\bot_\mathcal{H}})$ is confluent. But it is not normalising, because the term $\mathbf{\Omega}$, which is rootactive but not hypercollapsing, is not normalising.

4 Axioms of Closure under Expansion and Substitution

We now introduce and study some axioms involving closure under expansion and substitution.

▶ **Definition 4.1** (Closure under Expansion and Substitution). Let $\mathcal{U} \subseteq \mathcal{C}L^\infty$.

1. $\mathcal{U}$ satisfies the axiom of closure under w-expansion, if for all $M \in \mathcal{U}$ $N \in \mathcal{U}$ whenever $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow_w N$.
2. $\mathcal{U}$ satisfies the axiom of closure under substitution, if $M \in \mathcal{U}$ implies $M^\sigma \in \mathcal{U}$ for all $M \in \mathcal{C}L^\infty$ and substitutions σ from variables to terms in $\mathcal{C}L^\infty$.
3. We say that $\mathcal{U}$ satisfies the axiom of closure under $w\bot$-expansion from $\bot$, if for all $M \in \mathcal{C}L^\infty$, if $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow_{w\bot_\mathcal{U}} \bot$ then $M \in \mathcal{U}$.

▶ **Lemma 4.2.** *Let $\mathcal{U}$ be a subset of $\mathcal{C}L^\infty$. Then $\mathcal{U}$ satisfies both the axiom of indiscernibility and the axiom of closure under w-expansion if and only if satisfies the axiom of closure under $w\bot$-expansion from $\bot$.*

Proof. Assume $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow_{w\bot} \bot$ for some $M \in \mathcal{C}L^\infty$. Then $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow_w N \twoheadrightarrow\!\!\!\!\!\!\rightarrow_{\bot_\mathcal{U}} \bot$ for some $N \in \mathcal{C}L^\infty_\bot$ by postponement. We prove that $N \in \mathcal{U}$ by induction on the length β of $N \twoheadrightarrow\!\!\!\!\!\!\rightarrow_{\bot_\mathcal{U}} \bot$. Note that the last step should be a successor ordinal $\beta = \alpha + 1$. Hence, $N \twoheadrightarrow\!\!\!\!\!\!\rightarrow_\bot N' \rightarrow_\bot \bot$ for some N'. Let $N|_p$ denote the subterm of N at position p. For every position p of N' such that $N'|_p = \bot$, we have that $N|_p \twoheadrightarrow\!\!\!\!\!\!\rightarrow_\bot \bot$ is shorted than β. By induction hypothesis, $N|p \in \mathcal{U}$. By indiscernibility, $N \in \mathcal{U}$. Closure under w-expansion then gives us the desired $M \in \mathcal{U}$. For the converse, assume first that $M, N \in \mathcal{C}L^\infty$ such that $M \overset{\mathcal{U}}{\longleftrightarrow} N$. There exists P such that $P \preceq_\mathcal{U} M$ and $P \preceq_\mathcal{U} N$. If $M \in \mathcal{U}$ then $P \rightarrow_\bot \bot$. Hence, $N \twoheadrightarrow\!\!\!\!\!\!\rightarrow_\bot P \rightarrow_\bot \bot$. Since $\mathcal{U}$ satisfies closure under $w\bot$-expansion from $\bot$, we get $N \in \mathcal{U}$. Hence indiscernibility holds. Second, assuming that $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow_w N$ and $N \in \mathcal{U}$, we get $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow_w N \rightarrow_{\bot_\mathcal{U}} \bot$. By closure under $w\bot$-expansion from $\bot$ we can conclude $M \in \mathcal{U}$. Hence closure under w-reduction holds. ◀

▶ **Lemma 4.3.** *If a meaningless set $\mathcal{U} \subseteq \mathcal{C}L^\infty$ satisfies the axiom of closure under $w\bot$-expansion from $\bot$, then it also satisfies closure under substitution.*

Proof. Suppose $M \in \mathcal{U}$. Then $M \rightarrow_\bot \bot$. Because the reduction $\rightarrow_\bot$ is closed under substitution by definition we get $M^\sigma \rightarrow_\bot \bot$. But then $M^\sigma \in \mathcal{U}$ follows from closure under $w\bot$-expansion from $\bot$. ◀

▶ **Definition 4.4** (Set of $w\bot$-expansions from $\bot$). Let $\mathcal{U} \subseteq \mathcal{C}L^\infty$. We define $\overline{\mathcal{U}} = \{M \in \mathcal{C}L^\infty \mid M \twoheadrightarrow\!\!\!\!\!\!\rightarrow_{w\bot_\mathcal{U}} \bot\}$.

It is a immediate that $\overline{\mathcal{U}}$ satisfies closure under $w\bot$-expansion from $\bot$.

▶ **Lemma 4.5** ($\mathcal{U}$ and $\overline{\mathcal{U}}$ define the same reduction). *Let $\mathcal{U}$ be a meaningless set. We have that $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow_{w\bot_{\overline{\mathcal{U}}}} N$ iff $M \twoheadrightarrow\!\!\!\!\!\!\rightarrow_{w\bot_\mathcal{U}} N$.*

Proof. By induction on the length of the reduction sequence. We only show the case when the length is one. Let $M = C[P] \to_{\bot_{\overline{\mathcal{U}}}} C[\bot] = N$. Then, $P \preceq_{\overline{\mathcal{U}}} Q$ and $Q \in \overline{\mathcal{U}}$. By definition of $\overline{\mathcal{U}}$, we have that $Q \twoheadrightarrow\!\!\!\!\!\rightarrow_{w\bot_{\mathcal{U}}} \bot$. Since $P \preceq_{\overline{\mathcal{U}}} Q$, we have that P is obtained from Q by replacing some of its subterms in $\overline{\mathcal{U}}$ by $\bot$. Each of these subterms $w\bot_{\mathcal{U}}$-reduces to $\bot$ so that we can construct a reduction sequence $Q \twoheadrightarrow\!\!\!\!\!\rightarrow_{w\bot_{\mathcal{U}}} P$. By Confluence (Theorem 3.13), we have that $P \twoheadrightarrow\!\!\!\!\!\rightarrow_{w\bot_{\mathcal{U}}} \bot$. Hence, $C[P] \twoheadrightarrow\!\!\!\!\!\rightarrow_{w\bot_{\mathcal{U}}} C[\bot]$. The converse is trivial. ◀

As consequence of this lemma we have $\overline{\overline{\mathcal{U}}} = \overline{\mathcal{U}}$ and we obtain the corollary:

▶ **Corollary 4.6.** *If $\mathcal{U}$ is meaningless then $(\mathcal{CL}^{\infty}_{\bot}, \twoheadrightarrow\!\!\!\!\!\rightarrow_{w\bot_{\overline{\mathcal{U}}}})$ is confluent. Moreover, if $\mathcal{U}$ satisfies rootactiveness, then $(\mathcal{CL}^{\infty}_{\bot}, \twoheadrightarrow\!\!\!\!\!\rightarrow_{w\bot_{\overline{\mathcal{U}}}})$ is normalising.*

5 Meaningless Sets

In this section we will show that there are uncountably many meaningless sets in infinitary combinatory logic. This contrasts with the handful of meaningless sets (among which the rootactive terms) known for orthogonal infinitary term rewriting systems [8].

We will identify sets of meaningless terms that define the same reduction. Hence, by Lemma 4.5, we will consider those sets of meaningless terms that satisfy $\mathcal{U} = \overline{\mathcal{U}}$. Or, equivalently, using Lemma 4.2, we will restrict ourselves to sets of meaningless terms that satisfy the extra axiom of closure under w-expansion.

We will show that the class $\mathcal{L}_{CL}$ of sets of meaningless terms that satisfy w-expansion is a bounded lattice ordered by set inclusion. The meet (denoted by $\sqcap$) coincides with the intersection. The join $\mathcal{U} \sqcup \mathcal{V}$ is the least meaningless set containing $\mathcal{U} \cup \mathcal{V}$. By construction the set $\mathcal{CL}^{\infty}$ is the largest element of $\mathcal{L}_{CL}$ and $\mathcal{H}$ the smallest one.

The lattice $\mathcal{L}_{CL}$ has a somewhat richer structure than the corresponding lattice of meaningless sets of lambda calculus [16]. Below $\mathcal{CL}^{\infty}$ we find the unsolvables as the next largest set, as the notion of solvability for finite combinatory logic extends to $\mathcal{CL}^{\infty}$ and $\mathcal{CL}^{\infty}_{\bot}$ [1, 3].

▶ **Definition 5.1** (Solvable Term). Let $M \in \mathcal{CL}^{\infty}_{\bot}$.
1. A closed term M is *solvable* if there exist $P_1 \dots P_k$ such that $MP_1 \dots P_k \twoheadrightarrow_w \mathbf{I}$.
2. A term M is *solvable* if M^{σ} is solvable for some substitution σ replacing all variables in M by closed terms from $\mathcal{CL}^{\infty}$. Terms that are not solvable are called unsolvable.
3. $\mathcal{NS} = \{M \in \mathcal{CL}^{\infty} \mid M \text{ is not solvable}\}$.

We will first show that $\mathcal{NS}$ is indeed an element of $\mathcal{L}_{CL}$, before showing that $\mathcal{NS}$ is the largest element in $\mathcal{L}_{CL}$ below $\mathcal{CL}^{\infty}$. We need some useful lemmas to do so. The first lemma follows by induction on the length of the reduction sequence and it holds for any set $\mathcal{U}$.

▶ **Lemma 5.2.** *If $P \twoheadrightarrow\!\!\!\!\!\rightarrow_w P'$ and $P \preceq_{\mathcal{U}} M$ then $M \twoheadrightarrow\!\!\!\!\!\rightarrow_w M'$ and $P' \preceq_{\mathcal{U}} M'$ for some M'*

▶ **Lemma 5.3.** *If $Q \preceq_{\mathcal{NS}} N$ and $N \twoheadrightarrow_w \mathbf{I}$ then $Q \twoheadrightarrow_w \mathbf{I}$.*

Proof. By induction on the length n of the reduction sequence. If $n = 0$ then $Q \preceq_{\mathcal{NS}} N = \mathbf{I}$ and $Q = \mathbf{I}$. If $n > 0$ then we can only have that either $N = \mathbf{K}N_1 \dots N_k$ or $N = \mathbf{S}N_1 \dots N_{k+1}$ with $k \geq 2$ because the normal form of N is $\mathbf{I}$. We do the case $N = \mathbf{S}N_1 \dots N_k$ with $k \geq 3$ where the reduction sequence is $N = \mathbf{S}N_1 \dots N_k \to_w N_1N_3(N_2N_3)N_4 \dots N_k \twoheadrightarrow_w \mathbf{I}$. Hence N and all its prefixes $\mathbf{S}N_1 \dots N_i$ for $i \leq k$ are solvable. If $Q \preceq_{\mathcal{NS}} N$ then $Q = \mathbf{S}Q_1 \dots Q_k$ with $Q_i \preceq_{\mathcal{NS}} N_i$ for $1 \leq i \leq k$. By IH, $Q_1Q_3(Q_2Q_3)Q_4 \dots Q_k \twoheadrightarrow_w \mathbf{I}$. Hence $Q \twoheadrightarrow_w \mathbf{I}$. ◀

▶ **Lemma 5.4.** *Let $P \in \mathcal{CL}^{\infty}_{\bot}$ such that $P \preceq_{\mathcal{NS}} M$. Then, P is solvable iff M is solvable.*

Proof. Suppose P is solvable. Then there exist a substitution σ and terms $Q_1, \ldots, Q_n$ such that $P^\sigma Q_1 \ldots Q_n \twoheadrightarrow_w \mathbf{I}$. By Lemma 5.2, $M^\sigma Q_1 \ldots Q_n \twoheadrightarrow_w \mathbf{I}$. Hence, M is solvable.

Suppose M is solvable. Then, there exist a substitution σ and terms $Q_1, \ldots Q_n$ such that $M^\sigma Q_1 \ldots Q_n \twoheadrightarrow_w \mathbf{I}$. Assume that P is unsolvable. Then whenever $P^\sigma Q_1 \ldots Q_n \twoheadrightarrow_w Q$ it follows that $Q \neq \mathbf{I}$. Since $P^\sigma Q_1 \ldots Q_n \preceq_{\mathcal{NS}} M^\sigma Q_1 \ldots Q_n$ we get by Lemma 5.2 that $M^\sigma Q_1 \ldots Q_n \twoheadrightarrow_w N$ for some $Q \preceq_{\mathcal{NS}} N$. By confluence of $w\bot_{\mathcal{R}}$-reduction, we have that $N \twoheadrightarrow_{w\bot_{\mathcal{R}}} \mathbf{I}$. Hence by postponement we can factor this reduction into a w-reduction followed by a $\bot_{\mathcal{U}}$-reduction to $\mathbf{I}$. Because $\mathbf{I}$ does not contain any $\bot$, the latter must be necessarily must be of zero length. Hence $N \twoheadrightarrow_w \mathbf{I}$. Applying Lemma 5.3, we find that then also $Q \twoheadrightarrow_w \mathbf{I}$. Hence P is solvable as well. ◀

▶ **Lemma 5.5.** *If $M \in \mathcal{U}$ and $MP_1 \ldots P_n \twoheadrightarrow_w Q$ then $Q = NP_{i+1} \ldots P_n$ where $N \in \mathcal{U}$.*

Proof. This is proved by induction on the length of the reduction. It is enough to consider one step $MP_1 \ldots P_n \to_w Q$. If the w-redex is inside M or one of the P_i's, the claim is immediate. If the w-redex is of the form $MP_1 \ldots P_i$ for $i > 0$ then by overlap, we have that $MP_1 \ldots P_i \in \mathcal{U}$. By closure under reduction, the contracted redex is in $\mathcal{U}$. ◀

▶ **Theorem 5.6** (Meaningless Set $\mathcal{NS}$)**.** *The set $\mathcal{NS}$ is a meaningless set. Moreover, it is the largest meaningless set satisfying the axiom of closure under w-expansion which is a proper subset of $\mathcal{CL}^\infty$. In particular, $\mathcal{NS} = \overline{\mathcal{NS}} \subset \mathcal{CL}^\infty$.*

Proof. Let R be a rootactive term and σ a substitution. Then R^σ is also rootactive and $R^\sigma P_1 \ldots P_n$ can not reduce to $\mathbf{I}$. Therefore all rootactive terms are unsolvable and $\mathcal{NS}$ satisfies rootactiveness.

To prove closure under w-reduction, suppose M is unsolvable and $M \twoheadrightarrow_w N$. Assume N is solvable. Then $N^\sigma P_1 \ldots P_n \twoheadrightarrow_w \mathbf{I}$ for some σ, $P_1, \ldots, P_n$. Now M is solvable too, because $M^\sigma P_1 \ldots P_n \twoheadrightarrow_w N^\sigma P_1 \ldots P_n \twoheadrightarrow_w \mathbf{I}$. Contradiction. Hence N is unsolvable.

Overlap is also easy to prove. Suppose $\mathbf{S}MNP$ is solvable. Then there exist $\sigma, Q_1, \ldots, Q_n$ such that $(\mathbf{S}MNP)^\sigma Q_1 \ldots Q_n \twoheadrightarrow_w \mathbf{I}$. Hence, $\mathbf{S}$, $\mathbf{S}M$ and $\mathbf{S}MN$ are also solvable. Overlap with $\mathbf{K}$ goes similar.

We now prove indiscernibility. If $M \overset{\mathcal{NS}}{\longleftrightarrow} N$, then both $P \preceq_{\mathcal{NS}} M$ and $P \preceq_{\mathcal{NS}} N$ for some $P \in \mathcal{CL}^\infty_\bot$. It follows from Lemma 5.4 that $M \in \mathcal{NS}$ iff $N \in \mathcal{NS}$.

Next we prove that $\mathcal{NS}$ is closed under under w-expansion. Suppose $M \twoheadrightarrow_w N$. Assume M is solvable. If we show that N is solvable, then closure under w-reduction follows by contraposition. Solvability of M implies that $MP_1 \ldots P_n \twoheadrightarrow_w \mathbf{I}$ for some P_i. Applying Theorem 3.12 on confluence modulo hypercollapsing terms, we get $N \twoheadrightarrow_w \mathbf{I}$, as $\mathbf{I}$ does not contain any hypercollapsing subterms. Hence N is solvable.

Finally we prove that $\mathcal{NS}$ is the largest meaningless set closed under under w-expansion, which is a proper subset of $\mathcal{CL}^\infty$. Now, suppose that $M \in \mathcal{U}$ and $M \notin \mathcal{NS}$. Hence, there exists a substitution σ and terms $P_1, \ldots P_n$ such that $M^\sigma P_1 \ldots P_n \twoheadrightarrow_w \mathbf{I} = \mathbf{SKK}$. By Lemma 5.5, we have that either $\mathbf{S}$, $\mathbf{SK}$ or $\mathbf{SKK}$ are in $\mathcal{U}$. For any term N, we have that $\mathbf{SKK}N \twoheadrightarrow_w N$. By the axioms of overlap and closure under reduction, $N \in \mathcal{U}$. ◀

Note that a term in $\mathcal{CL}^\infty$ has either one of the following forms: (1) $\mathbf{S}$, $\mathbf{S}P$, $\mathbf{S}PQ$, $\mathbf{K}$, $\mathbf{K}P$ or $\mathbf{K}PQ$; (2) $xP_1 \ldots P_n$ for $n \geq 0$; (3) $((\ldots)P_2)P_1$; (4) $\mathbf{S}P_1 \ldots P_n$ for $n \geq 3$; (5) $\mathbf{K}P_1 \ldots P_n$ for $n \geq 2$. Having this in mind we give the following definition, reminiscent of the analogous Definition 4.1 in [16] for infinitary lambda calculus.

▶ **Definition 5.7** (Head Normal Form)**.** Let $M \in \mathcal{CL}^\infty_\bot$.

$$\begin{tikzcd}
& & \mathcal{R} \sqcup \mathcal{K}^\infty \sqcup \mathcal{S}^\infty \sqcup \mathcal{A}^\infty & & \\
& & & & \mathcal{R} \sqcup \mathcal{K}^\infty \sqcup \mathcal{S}^\infty \arrow[ull, Rightarrow] \\
\mathcal{R} \sqcup \mathcal{S}^\infty \sqcup \mathcal{A}^\infty \arrow[uurr] & & & & \\
& & & \mathcal{R} \sqcup \mathcal{S}^\infty \arrow[ulll, Rightarrow] \arrow[uur] & \\
& & \mathcal{R} \sqcup \mathcal{K}^\infty \sqcup \mathcal{A}^\infty \arrow[uuuu] & & \\
& & & & \mathcal{R} \sqcup \mathcal{K}^\infty \arrow[uuuu] \arrow[ull, Rightarrow] \\
\mathcal{NHF} = \mathcal{R} \sqcup \mathcal{A}^\infty \arrow[uuuu] \arrow[uurr] & & & & \\
& & & \mathcal{A}^{\mathcal{R}} \arrow[ulll, Rightarrow] \arrow[uuuu] \arrow[uur] & \\
& & \mathcal{R} \arrow[uull, Rightarrow] \arrow[ur, Rightarrow] & &
\end{tikzcd}$$

Figure 1 A fragment of $\mathcal{L}_{CL}$: single (double) arrows indicate that the corresponding open intervals are singletons (uncountable). A similar fragment can be built by replacing $\mathcal{R}$ by $\mathcal{H}$ in the above diagram.

1. M is in *head normal form (hnf or w-hnf)* if it is one of the forms $\mathbf{K}$, $\mathbf{K}P$, $\mathbf{S}$, $\mathbf{S}P$, $\mathbf{S}PQ$ or $xP_1 \dots P_n$.
2. $\mathcal{NHF} = \{M \in \mathcal{C}L^\infty \mid$ there is no N such that $M \twoheadrightarrow_w N$ and N is in hnf$\}$.

The set $\mathcal{NHF}$ is a set of meaningless terms that satisfies closure under w-expansion. Terms in $\mathcal{NHF}$ are exactly the opaque terms, i.e. their reducts cannot overlap any redex [8]. Unlike lambda calculus, terms with head normal forms do not correspond to solvable terms in combinatory logic. For example, the head normal form $\mathbf{K\Omega}$ is unsolvable in combinatory logic. The reason is that head normal forms in combinatory logic are related to weak head normal forms in lambda calculus: M is a hnf in combinatory logic iff M_λ is a weak head normal form in lambda calculus [4]. To define many more elements of $\mathcal{L}_{CL}$ we define the following forms and sets:

▶ **Definition 5.8.** **1.** $\mathcal{A}^Y_X = \{M \in \mathcal{C}L^\infty \mid M \twoheadrightarrow_w N$ and N is a X,Y-ha$\}$ where M is a *head active form relative to X and Y* (X,Y-ha) if $M = RP_1 \dots P_k$, $R \in Y$ and $P_i \in X$ for $1 \le i \le k$.

2. $\mathcal{A}^\infty_X = \{M \in \mathcal{C}L^\infty \mid M \twoheadrightarrow_w N$ and N is a X-il $\}$ where M is an *infinite left spine form relative to X* (X-il) if $M = (\dots P_2)P_1$ and $P_i \in X$ for all i.
3. $\mathcal{K}^\infty = \{M \in \mathcal{C}L^\infty \mid M \twoheadrightarrow_w \mathbf{K}^\omega\}$ where $\mathbf{K}^\omega = \mathbf{K}(\mathbf{K}(\mathbf{K}(\dots)))$.
4. $\mathcal{S}^\infty = \{M \in \mathcal{C}L^\infty \mid M \twoheadrightarrow_w \mathbf{S}^\omega\}$ where $\mathbf{S}^\omega = \mathbf{S}(\mathbf{S}(\mathbf{S}(\dots)))$.

When $X = \mathcal{C}L^\infty$ in $\mathcal{A}^Y_X$ or $\mathcal{A}^\infty_X$ we drop the phrase *relative to* X and the subscript X.

Using the $\mathcal{H}$ or $\mathcal{R}$ as a base we can make two sublattices of $\mathcal{L}_{CL}$ as depicted in Figure 1. The top elements of these fragments, $\mathcal{R} \sqcup \mathcal{K}^\infty \sqcup \mathcal{S}^\infty \sqcup \mathcal{A}^\infty$ and $\mathcal{H} \sqcup \mathcal{K}^\infty \sqcup \mathcal{S}^\infty \sqcup \mathcal{A}^\infty$ are proper subsets of $\mathcal{NS}$, because neither of the two sets contain the unsolvable $\mathbf{S}(\mathbf{K}(\mathbf{S}(\mathbf{K}(\dots))))$. Note that $\mathcal{NHF} = \mathcal{A}^{\mathcal{R}} \cup \mathcal{A}^\infty = \mathcal{R} \sqcup \mathcal{A}^\infty$. The intervals of Figure 1 that have uncountable cardinality, indicated by $\Rightarrow$, follow from Theorem 5.9 and the fact that $\mathsf{nf}_{\mathcal{NS}}(\mathcal{C}L^\infty) \cap \mathcal{C}L^\infty$ is uncountable.

▶ **Theorem 5.9** (Meaningless Sets). *Let $X \subseteq \mathsf{nf}_{\mathcal{NS}}(\mathcal{C}L^\infty) \cap \mathcal{C}L^\infty$ and $Y \in \{\mathcal{R}, \mathcal{H}\}$. The following sets are sets of meaningless terms that are closed under w-expansion: $\mathcal{A}^Y_X$, $\mathcal{A}^Y_X \cup \mathcal{A}^\infty_X$, $\mathcal{A}^Y \cup \mathcal{A}^\infty_X$, $\mathcal{A}^Y \cup \mathcal{K}^\infty \cup \mathcal{A}^\infty_X$, $\mathcal{A}^Y \cup \mathcal{S}^\infty \cup \mathcal{A}^\infty_X$ and $\mathcal{A}^Y \cup \mathcal{S}^\infty \cup \mathcal{K}^\infty \cup \mathcal{A}^\infty_X$.*

Proof. It is straightforward to show that these sets satisfy rootactiveness, overlap, closure under w-reduction and closure under w-expansion. We show that $\mathcal{U} = \mathcal{A}_X^{\mathcal{R}}$ satisfies indiscernibility for $X \subseteq \mathsf{nf}_{\mathcal{NS}}(\mathcal{CL}^\infty) \cap \mathcal{CL}^\infty$. If $M \overset{\mathcal{U}}{\longleftrightarrow} N$ then there exists P such that $P \preceq_{\mathcal{U}} M$ and $P \preceq_{\mathcal{U}} N$. By Lemma 5.2, we can assume that $P = \mathsf{skel}(P)$. We discuss cases according to the shape of P. The interesting case is when $P = \bot P_1 \ldots P_n$. Then $M = M_0 M_1 \ldots M_n$ and $N = N_0 N_1 \ldots N_n$. Since $X \subseteq \mathsf{nf}_{\mathcal{NS}}(\mathcal{CL}^\infty)$, we have that $P_i = M_i = N_i$ for all $1 \leq i \leq n$. It is clear that $M \in \mathcal{A}_X^{\mathcal{R}}$ iff $N \in \mathcal{A}_X^{\mathcal{R}}$. ◀

6 Application to Combinatory Algebras

Models for combinatory logic have a simple structure: they are *combinatory algebras*. Models for lambda calculus are λ-algebras which are combinatory algebras that satisfy some further properties. We show that there is an uncountable number of combinatory algebras that are not λ-algebras. We recall the definition of combinatory and λ-algebras from [1].

▶ **Definition 6.1** (Combinatory Algebra). A *combinatory algebra* is a structure $(X, \cdot, k, s)$ where $\cdot$ is a binary operation on X and $k, s \in X$ satisfy $kxy = x$ and $sxyz = xz(yz)$.

Given a valuation ρ mapping variables to terms in X we interpret terms of $\mathcal{CL}$ in a combinatory algebra $(X, \cdot, k, s)$ as follows: $[\![x]\!]_\rho = \rho(x)$, $[\![\mathbf{K}]\!]_\rho = k$, $[\![\mathbf{S}]\!]_\rho = s$ and $[\![MN]\!]_\rho = [\![M]\!]_\rho \cdot [\![N]\!]_\rho$.

▶ **Definition 6.2** (Lambda Algebra). A combinatory algebra is a λ-*algebra* if $M_\lambda =_\beta N_\lambda$ then $[\![M]\!]_\rho = [\![N]\!]_\rho$ for all $M, N \in \mathcal{CL}$.

▶ **Theorem 6.3** (Combinatory Algebra induced by Infinitary Combinatory Logic). *Let $\mathcal{U} \subset \mathcal{CL}^\infty$ be closed under $w\bot$-expansions from $\bot$. If $(\mathcal{CL}_\bot^\infty, \twoheadrightarrow_{w\bot_{\mathcal{U}}})$ is confluent and normalising, it induces a combinatory algebra that is not a λ-algebra which is given by $(\mathsf{nf}_{\mathcal{U}}(\mathcal{CL}^\infty), \cdot, \mathbf{K}, \mathbf{S})$ where $M \cdot N = \mathsf{nf}_{\mathcal{U}}(MN)$.*

Proof. Assume $\mathcal{U} \subset \mathcal{CL}^\infty$. A term is interpreted by its normal form. In a λ-algebra, the term $\mathbf{K}$ should be equal to $\mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})\mathbf{K}))(\mathbf{K}(\mathbf{S}(\mathbf{KK}))$. Since these two terms are in w-normal form, they will have the same $w\bot_{\mathcal{U}}$-normal form only if they both reduce to $\bot$. This is not possible, as then $\mathbf{K}$ should belong to $\mathcal{U}$, which would imply by Lemma 7.2 that $\mathcal{U} = \mathcal{CL}^\infty$. Contradiction. ◀

▶ **Corollary 6.4** (Uncountable Combinatory Algebras). *There is an uncountable number of combinatory algebras that are not λ-algebras.*

Proof. By Theorem 5.9, there are uncountably many meaningless sets satisfying rootactiveness. Each of them gives rise to a different, confluent and normalising infinitary $\bot$-extension of $\mathcal{CL}$ by Theorems 5.9 and 3.16. By Theorem 6.3 none of the induced combinatory algebras is a λ-model. ◀

7 Weakening the Axioms of Meaningless Terms

In this section, we first show that confluence implies hypercollapseness, closure under w-reduction, indiscernibility and a weaker form of overlap. We can prove the converse only under the extra condition of rootactiveness.

To weaken the axiom of overlap, we inspect when overlap between $\perp$-reduction and w-reduction occurs. Overlap happens when the $\perp$-redex is of the form $\mathbf{K}M$, $\mathbf{S}M$ or $\mathbf{S}MN$. This gives a divergence that can be resolved with the axiom of overlap, e.g.

$$\begin{tikzcd}
& \mathbf{S}MNP \arrow[dl, "\perp"'] \arrow[dr, "w"] & \\
\perp P \arrow[dr, dotted, "\perp"'] & & MP(NP) \arrow[dl, dotted, "\perp"'] \\
& \perp &
\end{tikzcd}$$

There is, however, another way of resolving this divergence. Suppose that $M = (\mathbf{K}W)$ for some $W \in \mathcal{U}$ and $N = \mathbf{I}$. Then, we have that $\mathbf{S}(\mathbf{K}W)\mathbf{I}P \twoheadrightarrow_w WP \to_\perp \perp$. This is not the only alternative to resolve the divergence. Suppose $M = \mathbf{S}(\mathbf{KK})W$ for some $W \in \mathcal{U}$.. Then, we have that $\mathbf{S}(\mathbf{S}(\mathbf{KK})W)NP \twoheadrightarrow_w WP \to_\perp \perp$. In general, there is an alternative resolution of the divergence whenever there exists a $W \in \mathcal{U}$ such that $MP(NP) \twoheadrightarrow\!\!\!\!\rightarrow_w WP$.

▶ **Definition 7.1** (Weak Overlap). Let $\mathcal{U} \subseteq \mathcal{CL}^\infty$. We say that $\mathcal{U}$ satisfies the axiom of weak overlap if we have that for all $M, N, P \in \mathcal{CL}^\infty$:
1. If $\mathbf{K}$ or $\mathbf{K}M \in \mathcal{U}$ then $\mathbf{K}MN \in \mathcal{U}$.
2. If $\mathbf{S}$ or $\mathbf{S}M \in \mathcal{U}$ then $\mathbf{S}MN \in \mathcal{U}$ or $\mathbf{S}MNP \in \mathcal{U}$.
3. If $\mathbf{S}MN \in \mathcal{U}$ then $\mathbf{S}MNP \in \mathcal{U} \cup \mathcal{R}$ or $\mathbf{S}MNP \to_w MP(NP) \twoheadrightarrow\!\!\!\!\rightarrow_w WP$ and $W \in \mathcal{U} \cap (\mathcal{A}^\mathcal{R} \cup \mathcal{A}^\infty)$.

Clearly, overlap implies weak overlap.

▶ **Lemma 7.2** ($\mathbf{K}$ or $\mathbf{S}$ in $\mathcal{U}$ implies all Terms are in $\mathcal{U}$). *Let $\mathcal{U} \subseteq \mathcal{CL}^\infty$ satisfy closure under $w\perp$-expansion from $\perp$ and $(\mathcal{CL}^\infty_\perp, \twoheadrightarrow\!\!\!\!\rightarrow_{w\perp_\mathcal{U}})$ be confluent. If $\mathbf{K} \in \mathcal{U}$ or $\mathbf{S} \in \mathcal{U}$ then $\mathcal{U} = \mathcal{CL}^\infty$.*

Proof. Suppose that $\mathbf{K} \in \mathcal{U}$. $\mathbf{K}xy$ reduces to $\perp xy$ and to x. By confluence, these two terms should have a common reduct. The only possibility is that this reduct is $\perp$. By closure under $w\perp$-expansion from $\perp$, $x \in \mathcal{U}$. By Lemma 4.3, $M \in \mathcal{U}$ for all $M \in \mathcal{CL}^\infty$. Suppose now that $\mathbf{S} \in \mathcal{U}$. We have that $\mathbf{SK}xy \twoheadrightarrow_w y$ and $\mathbf{SK}xy \to_\perp \perp\mathbf{K}xy$. By confluence, y and $\perp\mathbf{K}xy$ have a common reduct Q. The only possibility is that $Q = \perp$. Hence, $y \in \mathcal{U}$. By Lemma 4.3, $\mathcal{U}$ satisfies closure under substitution and hence, $M \in \mathcal{U}$ for all $M \in \mathcal{CL}^\infty$. ◀

The next lemma is proved by induction on the length of the reduction sequence.

▶ **Lemma 7.3.** *If $M \twoheadrightarrow\!\!\!\!\rightarrow_w N$ and all occurrences of x in M are in a term of the form xy then so are all occurrences of x in N.*

▶ **Theorem 7.4** (Necessary Conditions for Confluence). *Let $\mathcal{U} \subseteq \mathcal{CL}^\infty$ be closed under $w\perp$-expansion from $\perp$. If $(\mathcal{CL}^\infty_\perp, \twoheadrightarrow\!\!\!\!\rightarrow_{w\perp_\mathcal{U}})$ is confluent, then $\mathcal{U}$ satisfies the axioms of hypercollapseness, closure under w-reduction, indiscernibility and weak overlap.*

Proof. If $\mathcal{U} = \mathcal{CL}^\infty$ then it satisfies all the axioms. From now on, suppose that $\mathcal{U} \subsetneq \mathcal{CL}^\infty$. We have that $\mathcal{U}$ satisfies indiscernibility by Lemma 4.2.

We now prove that $\mathcal{U}$ satisfies closure under w-reduction as follows. Suppose $M \twoheadrightarrow\!\!\!\!\rightarrow_w N$ and $M \in \mathcal{U}$. It follows from $M \to_\perp \perp$ and confluence that $N \twoheadrightarrow\!\!\!\!\rightarrow_{w\perp_\mathcal{U}} \perp$. By closure under $w\perp$-expansion from $\perp$, we have that $N \in \mathcal{U}$.

We prove that $\mathcal{U}$ satisfies hypercollapseness by showing that all hypercollapsing terms reduce to $\perp$. The infinite term $\mathbf{C}_x[\mathbf{C}_y[\mathbf{C}_x[\mathbf{C}_y[\ldots]]]]$ can reduce to both $\mathbf{C}_x^\omega$ and $\mathbf{C}_y^\omega$. By confluence, both $\mathbf{C}_x^\omega$ and $\mathbf{C}_y^\omega$ reduce to some common term Q. By postponement, $\mathbf{C}_x^\omega \twoheadrightarrow\!\!\!\!\rightarrow_\perp Q$

and $\mathbf{C}_y^\omega \twoheadrightarrow\!\!\!\!\rightarrow_\perp Q$ because $\mathbf{C}_x^\omega$ and $\mathbf{C}_y^\omega$ can only ω-reduce to themselves. Since $\mathbf{C}_x$ and $\mathbf{C}_y$ have no prefix in common, Q should be $\perp$.

Now suppose P is an arbitrary hypercollapsing term. Then, we have $P \twoheadrightarrow\!\!\!\!\rightarrow \mathbf{C}_{M_1}[\mathbf{C}_{M_2}[\ldots]]$ where $\mathbf{C}_{M_i}[\;] = (\mathbf{K}[\;]M_i)$ for some collapsing tower $\mathbf{C}_{M_1}[\mathbf{C}_{M_2}[\ldots]]$. The infinite collapsing tower $\mathbf{C}_{M_1}[\mathbf{C}_x[\mathbf{C}_{M_2}[\mathbf{C}_x[\ldots]]]]$ can reduce both to $\mathbf{C}_{M_1}[\mathbf{C}_{M_2}[\ldots]]$ and $\mathbf{C}_x[\mathbf{C}_x[\ldots]]$. Since $\mathbf{C}_x[\mathbf{C}_x[\ldots]]$ reduces to $\perp$, we find that by confluence, also $\mathbf{C}_{M_1}[\mathbf{C}_{M_2}[\ldots]]$ should reduce to $\perp$. Hence, all the hypercollapsing terms reduce to $\perp$. By closure under $\omega\perp$-expansion from $\perp$, we have that $\mathcal{H} \subseteq \mathcal{U}$.

We prove that $\mathcal{U}$ satisfies weak overlap by proving the clauses of Definition 7.1.

1. If $\mathbf{K} \in \mathcal{U}$ then by Lemma 7.2, we would have $\mathcal{U} = \mathcal{CL}^\infty$. Assume $\mathbf{K}M \in \mathcal{U}$. Then, $\mathbf{K}MN \to_w M$ and $\mathbf{K}MN \to_\perp \perp N$ for all N. Hence by confluence M and $\perp N$ must have a common reduct for any N. Hence in particular $\perp x$ and $\perp y$ must have a common reduct, which can only be $\perp$. Hence also M reduces to $\perp$ and therefore we see that $\mathbf{K}MN \in \mathcal{U}$ by closure under w-expansion.
2. If $\mathbf{S} \in \mathcal{U}$ then by Lemma 7.2 we would have $\mathcal{U} = \mathcal{CL}^\infty$. Suppose $\mathbf{S}M \in \mathcal{U}$ and x, y do not occur in M. Then, $\mathbf{S}Mxy$ reduces to $\perp xy$ and to $My(xy)$. They should both have a common reduct N. We have three cases. The first case is when $N = \perp xy$. Then, $My(xy) \twoheadrightarrow\!\!\!\!\rightarrow_w Wxy$ for some $W \in \mathcal{U}$. By Lemma 7.3, this is not possible. The second case is when $N = \perp y$. By closure under $w\perp$-expansion from $\perp$, $\mathbf{S}Mx \in \mathcal{U}$. By Lemma 4.3, $\mathbf{S}MN \in \mathcal{U}$ for all $N \in \mathcal{U}$. The third case is when $N = \perp$. By closure under $w\perp$-expansion from $\perp$, $\mathbf{S}Mxy \in \mathcal{U}$. By Lemma 4.3, $\mathbf{S}MNP \in \mathcal{U}$ for all $N, P \in \mathcal{CL}^\infty$.
3. By closure under substitutions (Lemma 4.3), it is enough to consider $P = x$. Assume $\mathbf{S}MN \in \mathcal{U}$ and $\mathbf{S}MNx \notin \mathcal{U}$. By confluence, $\perp x$ and $\mathsf{skel}(\mathbf{S}MNx)$ must have a common reduct. This common reduct cannot be $\perp$ because $\mathbf{S}MNx \notin \mathcal{U}$. Then, the common reduct should be $\perp x$, i.e. $\mathsf{skel}(\mathbf{S}MNx) \twoheadrightarrow\!\!\!\!\rightarrow_{w\perp_\mathcal{U}} \perp x$. By postponement, $\mathsf{skel}(\mathbf{S}MNx) \twoheadrightarrow\!\!\!\!\rightarrow_w Q \twoheadrightarrow\!\!\!\!\rightarrow_\perp \perp x$. Hence, $Q = Wx$ and $W \twoheadrightarrow\!\!\!\!\rightarrow_\perp \perp$. By closure under expansion from $\perp$, $W \in \mathcal{U}$. Suppose $\mathsf{skel}(\mathbf{S}MNx)$ is not rootactive, then it has one of the following forms:

   a. $\mathsf{skel}(\mathbf{S}MNx)$ is $yP_1 \ldots P_k$. Then, $P_k = x$ and $yP_1 \ldots P_{k-1} \in \mathcal{U}$. It is not difficult to show that $\mathcal{U} = \mathcal{CL}^\infty$.
   b. $\mathsf{skel}(\mathbf{S}MNx)$ is $\mathbf{K}$ or $\mathbf{S}$. This case is impossible because $\mathbf{K}$ or $\mathbf{S}$ cannot reduce to $\perp x$.
   c. $\mathsf{skel}(\mathbf{S}MNx)$ is $\mathbf{K}P$ or $\mathbf{S}P$. Then $P = x$ and either $W \in \mathcal{U}$ is $\mathbf{K}$ or $\mathbf{S}$. By Lemma 7.2 we would have $\mathcal{U} = \mathcal{CL}^\infty$.
   d. $\mathsf{skel}(\mathbf{S}MNx)$ is $\mathbf{S}P_1P_2$. Then, $P_2 = x$ and $\mathbf{S}P_1 \twoheadrightarrow\!\!\!\!\rightarrow_w W$. By closure under w-expansion, $\mathbf{S}P_1 \in \mathcal{U}$. Similarly to case 3), we have that $\mathbf{S}P_1x \in \mathcal{U}$ and hence, $\mathbf{S}MNx \twoheadrightarrow\!\!\!\!\rightarrow_{w\perp_\mathcal{U}} \perp$. By closure under $w\perp$-expansion from $\perp$, $\mathbf{S}MNx \in \mathcal{U}$. This is a contradiction.
   e. $\mathsf{skel}(\mathbf{S}MNx)$ is either a head active form or a infinite left spine. So is W. Hence, $W \in \mathcal{U} \cap (\mathcal{A}^\mathcal{R} \cup \mathcal{A}^\infty)$.

◀

We do not know whether it is possible to prove the converse. However, we can prove that under the extra condition of rootactiveness the conditions necessary for confluence are also sufficient for confluence. The following lemma plays a crucial role in the proof of this result. In a similar scenario for lambda calculus this role was played by Lemma 5.5 in [16].

▶ **Lemma 7.5.** *Let $\mathcal{U} \subseteq \mathcal{CL}^\infty$ satisfy rootactiveness, closure under w-reduction, indiscernibility and weak overlap. If $M \twoheadrightarrow\!\!\!\!\rightarrow_{\perp_\mathcal{U}} N$ and N is a $w\perp_\mathcal{U}$-normal form, then $\mathsf{nf}_\mathcal{R}(M) \twoheadrightarrow\!\!\!\!\rightarrow_{\perp_\mathcal{U}} N$.*

Proof. We can suppose that $\mathcal{U} \neq \mathcal{C}L^\infty$ and $M \twoheadrightarrow^{out}_{\bot_\mathcal{U}} N$. Then, M is obtained from N by replacing some $\bot$'s by terms in $\mathcal{U}_\bot$. We consider the set of terms

$$\mathcal{W} = \{W \mid W \text{ is a maximal subterm of } M \text{ such that } W \in \mathcal{U}_\bot\}$$

Each subterm $W \in \mathcal{W}$ of M will be replaced by $\mathsf{nf}_\mathcal{R}(W)$ in depth-first leftmost order. We obtain a term M_0 such that $M \twoheadrightarrow_{w\bot_\mathcal{R}} M_0$. We have that $W \twoheadrightarrow_w W_0 \twoheadrightarrow_{\bot_\mathcal{R}} \mathsf{nf}_\mathcal{R}(W)$. By closure under w-reduction, $W_0 \in \mathcal{U}$. By rootactiveness, $\mathsf{nf}_\mathcal{R}(W) \preceq_\mathcal{U} W_0$. By definition of $\bot$, $\mathsf{nf}_\mathcal{R}(W) \to_{\bot_\mathcal{U}} \bot$. Hence, $M_0 \twoheadrightarrow_{\bot_\mathcal{U}} N$. Since N is a $w\bot_\mathcal{U}$-normal form, if an w-redex occurs in M_0 then it either occurs in some subterm $W \in \mathcal{W}$ or it overlaps some subterm $W \in \mathcal{W}$. By replacing W by $\mathsf{nf}_\mathcal{R}(W)$, we remove all the possible w-redexes that are in W. We may still have w-redexes that overlap a term in $\mathcal{U}$. Suppose $\mathsf{nf}_\mathcal{R}(W)$ overlaps a w-redex of the form $\mathsf{nf}_\mathcal{R}(W)N_1 \dots N_k$ in M_0. Then $WN_1 \dots N_k$ is a w-redex in M. We have that $\mathsf{nf}_\mathcal{R}(W)N_1 \dots N_k \notin \mathcal{U}_\bot$ because otherwise by indiscernibility $WN_1 \dots N_k \in \mathcal{U}_\bot$ and N would not be in $w\bot_\mathcal{U}$-normal form. By weak overlap, $\mathsf{nf}_\mathcal{R}(W)$ cannot be $\mathbf{K}$, $\mathbf{S}$, $\mathbf{K}P$ or $\mathbf{S}P$. By weak overlap, the only possibility is that $\mathsf{nf}_\mathcal{R}(W) = \mathbf{S}PQ$, $k = 1$ and $\mathbf{S}PQN_1 \twoheadrightarrow_w W_1N_1$ where $W_1 \in \mathcal{U} \cap (\mathcal{A}^\mathcal{R} \cup \mathcal{A}^\infty)$. In this case, to remove the w-redex, we replace the term W in M by $\mathsf{nf}_\mathcal{R}(W_1)$ instead of replacing it by $\mathsf{nf}_\mathcal{R}(W)$. The fact that W is either a head active term or an infinite left spine ensures that the normal form of WP can be calculated as the application of the normal forms of W and P. In other words, we have that

$$\mathsf{nf}_\mathcal{R}(WN_1) = \mathsf{nf}_\mathcal{R}(W_1)\mathsf{nf}_\mathcal{R}(N_1)$$

Since $\mathsf{nf}_\mathcal{R}(W_1) \preceq_\mathcal{R} W_1$, we have that $\mathsf{nf}_\mathcal{R}(W_1) \to_{\bot_\mathcal{U}} \bot$.

By replacing all terms in $\mathcal{W}$ in the fashion described above, we obtain a term M_1 in $w\bot_\mathcal{R}$-normal form such that $M \twoheadrightarrow_{w\bot_\mathcal{R}} M_0 \twoheadrightarrow_{w\bot_\mathcal{R}} M_1$ and $M_1 \twoheadrightarrow_{\bot_\mathcal{U}} N$. By confluence of $w\bot_\mathcal{R}$-reduction, we have that $M_1 = \mathsf{nf}_\mathcal{R}(M)$. ◀

▶ **Theorem 7.6** (Confluence). *Let $\mathcal{U} \subseteq \mathcal{C}L^\infty$. If $\mathcal{U}$ is satisfies rootactiveness, closure under w-reduction, indiscernibility and weak overlap then $(\mathcal{C}L^\infty_\bot, \twoheadrightarrow_{w\bot_\mathcal{U}})$ is confluent.*

Proof. The proof is described in the following diagram.

$$\begin{tikzcd}
 & & M \arrow[dll, two heads, "w\bot_\mathcal{U}"'] \arrow[dl, two heads, "w"'] \arrow[dr, two heads, "w"] \arrow[drr, two heads, "w\bot_\mathcal{U}"] & & \\
M_1 \arrow[r, phantom, "(1)"] \arrow[dd, two heads, "w\bot_\mathcal{U}"'] & L_1 \arrow[rr, phantom, "(2)"] \arrow[ddl, two heads, "\bot_\mathcal{U}"'] \arrow[dr, two heads, "w\bot_\mathcal{R}"] & & L_2 \arrow[r, phantom, "(1)"] \arrow[dl, two heads, "w\bot_\mathcal{R}"'] \arrow[ddr, two heads, "\bot_\mathcal{U}"] & M_2 \arrow[dd, two heads, "w\bot_\mathcal{U}"] \\
 & & \mathsf{nf}_\mathcal{R}(M) \arrow[ll, phantom, "(3)"] \arrow[rr, phantom, "(3)"] \arrow[dll, two heads, "\bot_\mathcal{U}"] \arrow[drr, two heads, "\bot_\mathcal{U}"'] \arrow[dd, phantom, "(4)"] & & \\
N_1 \arrow[drr, equal, "\bot_\mathcal{U}"'] & & & & N_2 \arrow[dll, equal, "\bot_\mathcal{U}"] \\
 & & N_1 = N_2 & &
\end{tikzcd}$$

Suppose we have a divergence $M_1 \twoheadleftarrow_{w\bot} M \twoheadrightarrow_{w\bot} M_2$. By rootactiveness for $\mathcal{U}$, we can reduce M_1 and M_2 further to their respective $w\bot_\mathcal{U}$-normal forms N_1 and N_2 by Theorem 3.15. (1) By closure under substitution for $\mathcal{U}$ and Theorem 3.14 we find L_1 and L_2 such that $M \twoheadrightarrow_w L_1 \twoheadrightarrow_{\bot_\mathcal{U}} N_1$ and $M \twoheadrightarrow_w L_2 \twoheadrightarrow_{\bot_\mathcal{U}} N_2$. (2) By Theorems 3.16 and 3.15, we construct the reductions $L_1 \twoheadrightarrow_{w\bot_\mathcal{U}} \mathsf{nf}_\mathcal{R}(M)$ and $L_2 \twoheadrightarrow_{w\bot_\mathcal{U}} \mathsf{nf}_\mathcal{R}(M)$. (3) By Lemma 7.5 we then find the reductions $\mathsf{nf}_\mathcal{R}(L_1) \twoheadrightarrow_{\bot_\mathcal{U}} N_1$ and $\mathsf{nf}_\mathcal{R}(L_2) \twoheadrightarrow_{\bot_\mathcal{U}} N_2$. By normalisation and confluence of $(\mathcal{C}L^\infty, \twoheadrightarrow_{w\bot_\mathcal{R}})$, we have $\mathsf{nf}_\mathcal{R}(M) = \mathsf{nf}_\mathcal{R}(L_1) = \mathsf{nf}_\mathcal{R}(L_2)$. (4) Finally Theorem 3.13 on confluence of $\bot_\mathcal{U}$ and the fact that N_1 and N_2 are by construction normal forms for $\bot_\mathcal{U}$-reduction implies that N_1 and N_2 are identical. ◀

▶ **Example 7.7.** Let $\mathcal{U} \subseteq \mathcal{C}L^\infty$. We construct the following sets:

$$\begin{aligned}\mathcal{U}^* &= \mathcal{U} \cup \{M \in \mathcal{C}L^\infty \mid M \twoheadrightarrow_w \mathbf{S}(\mathbf{K}W)\mathbf{I} \text{ and } W \in \mathcal{U}\}\\ \mathcal{U}^{**} &= \mathcal{U} \cup \{M \in \mathcal{C}L^\infty \mid M \twoheadrightarrow_w \mathbf{S}(\mathbf{S}(\mathbf{KK})W)N, W \in \mathcal{U} \text{ and } N \in \mathcal{C}L^\infty\}\end{aligned}$$

If $\mathcal{U}$ is a set of meaningless terms that contains $\mathcal{R}$ and it is closed under w-expansion, then $\mathcal{U}^*$ and $\mathcal{U}^{**}$ satisfy rootactiveness, indiscernibility, closure under w-reduction, w-expansion and weak overlap. They do not satisfy overlap and hence, they are not meaningless. Yet the corresponding infinitary combinatory logics are confluent and normalising by Theorems 7.6 and 3.15. There are uncountably many such sets.
The constructions $\mathcal{U}^*$ and $\mathcal{U}^{**}$ are related to the set $\mathcal{U}^\eta$ of weakly meaningless terms defined for infinitary lambda calculus in [16] where $\mathcal{U}^\eta = \mathcal{U} \cup \{M \in \Lambda^\infty \mid M \twoheadrightarrow_\beta \lambda x.Wx \text{ and } W \in \mathcal{U}\}$. It is easy to see that $(\mathbf{S}(\mathbf{K}W)\mathbf{I})_\lambda$ and $(\mathbf{S}(\mathbf{S}(\mathbf{KK})W)N)_\lambda$ both β-reduce to $\lambda x.W_\lambda x$.

▶ Remark. Note that Theorem 7.6 could not be proved using the schema of the proof of Theorem 3.18 since commutation of $\bot$-reduction and w-reduction outside $\mathcal{U}$ may not hold. To see this, take $\mathcal{R}^{**}$ defined in Example 7.7 and $M = (\mathbf{S}(\mathbf{KK})W)$. Then, the only way to join $\bot P \leftarrow_{\bot_\mathcal{U}} \mathbf{S}MNP \rightarrow_w MP(NP)$ is by $w\bot$-reducing $MP(NP)$ to $\bot P$.

8 Related and Future Work

Sufficient and Necessary Condition for Confluence. In [16], we define a notion of weak meaningless set for infinitary lambda calculus and prove that this is a sufficient and necessary condition for confluence of $\beta\bot$-reduction . In the case of infinitary combinatory logic, it remains open to give a sufficient and necessary condition for confluence of $w\bot$-reduction. We think that hypercollapseness, closure under w-reduction, weak overlap and indiscernibility should be sufficient condition for confluence besides of being necessary (Theorem 7.4) but also sufficient. In other words, it remains open to prove Theorem 7.6 assuming only hypercollapseness instead of rootactiveness.

Sufficient and Necessary Condition for Normalisation. In [16] we show that normalisation implies rootactiveness for infinitary lambda calculus with $\beta\bot$-reduction. This does not hold in the setting of combinatory logic with $w\bot$-reduction as witness by the following example of normalising infinitary combinatory logic that does not satisfy rootactiveness. Let $\mathbf{D}[\ ] = \mathbf{S}(\mathbf{K}[\ ])\mathbf{I}$. Note that $\mathbf{D}^\omega P$, the infinite nesting of such contexts applied to P is rootactive and reduces to itself for all P. Define:

$$\mathcal{D} = \{M \in \mathcal{R} \mid \forall P \in \mathcal{C}L^\infty \cdot M \not\twoheadrightarrow_w \mathbf{D}^\omega P\} \cup \{M \in \mathcal{C}L^\infty \mid M \twoheadrightarrow_w \mathbf{D}^\omega\}.$$

To construct the normal form of a term $M \in \mathcal{C}L^\infty_\bot$ for $w\bot_\mathcal{D}$-reduction we first $\rightarrow_w$-reduce M to its skeleton N. Next in the depth-first left-most order, we replace rootactive subterms that can not reduce to a term of the form $\mathbf{D}^\omega P$ by $\bot$; and we replace rootactive subterms that reduce to a term of the form $\mathbf{D}^\omega P$ by $\bot P$. Then we repeat the procedure *ad infinitum* on these latter terms P that still may contain untreated rootactive terms. The resulting reduction is strongly converging and its limit is a normal form in $\mathcal{C}L^\infty_{\bot_\mathcal{U}}$.

It remains open to find a sufficient and necessary condition for having a normalising infinitary combinatory logic $(\mathcal{C}L^\infty_\bot, \twoheadrightarrow_{w\bot_\mathcal{U}})$. This condition is necessarily weaker than rootactiveness.

Generalisation to Infinitary Term Rewriting and Combinatory Reduction Systems. A further next step following the explorations in lambda calculus and combinatory logic would be to investigate sufficient and necessary conditions for having confluence and normalisation in the wider context of orthogonal term rewriting systems and combinatory reduction systems.

Combinatory Algebras. Bethke, Klop and de Vrijer show that not every partial combinatory algebra can be completed [2]. We could define a partial combinatory algebra from the set of w-normal forms in $\mathcal{C}L^\infty$. The interpretation of a term M is N if $M \twoheadrightarrow_w N$ and N in w-normal form. The fact that this is a partial combinatory algebra follows from the normal form property [6]. This partial combinatory algebra is made complete by adding $\perp$. Since there are many sets of meaningless terms, we have many ways of completing it.

Selinger shows that the standard term algebra as a combinatory algebra cannot be ordered, i.e. every compatible partial order on it is trivial [13]. Lusin and Salibra show that there exists a wide class of combinatory algebras that admit extensions with a non-trivial compatible partial order [10]. Salibra shows that there is a continuum of unorderable lambda models [12]. In [14] we study orderability on the lambda models induced by the infinitary lambda calculus. It will also be interesting to study orderability on the combinatory algebras induced by the infinitary combinatory logic. This may shed new light on Plotkin's conjecture saying that an absolutely unorderable combinatory algebra exists, i.e. it cannot be embedded in any combinatory algebra admitting a non-trivial partial order [11].

Acknowledgements

We would like to thank the reviewers for detailed comments and helpful suggestions.

References

1 H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland, Amsterdam, Revised edition, 1984.

2 I. Bethke, J. W. Klop, and R. C. de Vrijer. Extending partial combinatory algebras. *Mathematical Structures in Computer Science*, 9(4):483–505, 1999.

3 J.R. Hindley and Seldin J.P. *Introduction to Combinators and λ-calculus.* Cambridge University Press, 1988.

4 B. Intrigila and R. Statman. Solution to the range problem for combinatory logic. *Fundamenta Informaticae*, 111(2):203—-222, 2011.

5 J.R. Kennaway and F.J. de Vries. Infinitary rewriting. In Terese, editor, *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*, pages 668–711. Cambridge University Press, 2003.

6 J.R. Kennaway, J.W. Klop, M.R. Sleep, and F.J. de Vries. Transfinite reductions in orthogonal term rewriting systems. *Information and Computation*, 119(1):18–38, 1995.

7 J.R. Kennaway, J.W. Klop, M.R. Sleep, and F.J. de Vries. Infinitary lambda calculus. *Theoretical Computer Science*, 175(1):93–125, 1997.

8 J.R. Kennaway, V. van Oostrom, and F.J. de Vries. Meaningless terms in rewriting. *Journal of Functional and Logic Programming*, Article 1:35 pp, 1999.

9 J.W. Klop and R. de Vrijer. First-order term rewriting systems. In Terese, editor, *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*, pages 24–59. Cambridge University Press, 2003.

10 S. Lusin and A. Salibra. A note on absolutely unorderable combinatory algebras. *J. Log. Comput.*, 13(4):481–502, 2003.

11 G. D. Plotkin. On a question of H. Friedman. *Inf. Comput.*, 126(1):74–77, 1996.

12 A. Salibra. Topological incompleteness and order incompleteness of the lambda calculu. *ACM Trans. Comput. Log.*, 4(3):379–401, 2003.

13 P. Selinger. Order-incompleteness and finite lambda reduction models. *Theor. Comput. Sci.*, 309(1-3):43–63, 2003.

14 P.G. Severi and F.J. de Vries. Order Structures for Böhm-like models. In *CSL*, volume 3634 of *LNCS*, pages 103–116. Springer, 2005.

15 P.G. Severi and F.J. de Vries. Decomposition and cardinality of intervals in the lattice of meaningless sets. In *WOLLIC*, volume 6642 of *LNAI*, pages 210–227. Springer, 2011.

16 P.G. Severi and F.J. de Vries. Weakening the Axiom of Overlap in the Infinitary Lambda Calculus. In *RTA*, volume 10, pages 313–328. LIPIcs, 2011.

17 J. Tromp. Kolmogorov complexity in combinatory logic, 1999. `http://homepages.cwi.nl/~tromp/cl/cl.html`.

A Rewriting Framework for Activities Subject to Regulations

Max Kanovich[1], Tajana Ban Kirigin[2], Vivek Nigam[3], Andre Scedrov[4], Carolyn Talcott[5], and Ranko Perovic[6]

1 **Queen Mary, University of London, UK, e-mail:** `mik@eecs.qmul.ac.uk`
2 **University of Rijeka, HR, e-mail:** `bank@math.uniri.hr`
3 **Ludwig-Maximilians-Universität, Germany, e-mail:** `vivek.nigam@ifi.lmu.de`
4 **University of Pennsylvania, USA, e-mail:** `scedrov@math.upenn.edu`
5 **SRI International, USA, e-mail:** `clt@csl.sri.com`
6 **Senior Clinical Trial Specialist, e-mail:** `perovicrankomd@gmail.com`

Abstract

Activities such as clinical investigations or financial processes are subject to regulations to ensure quality of results and avoid negative consequences. Regulations may be imposed by multiple governmental agencies as well as by institutional policies and protocols. Due to the complexity of both regulations and activities there is great potential for violation due to human error, misunderstanding, or even intent. Executable formal models of regulations, protocols, and activities can form the foundation for automated assistants to aid planning, monitoring, and compliance checking. We propose a model based on multiset rewriting where time is discrete and is specified by timestamps attached to facts. Actions, as well as initial, goal and critical states may be constrained by means of relative time constraints. Moreover, actions may have non-deterministic effects, *i.e.*, they may have different outcomes whenever applied. We demonstrate how specifications in our model can be straightforwardly mapped to the rewriting logic language Maude, and how one can use existing techniques to improve performance. Finally, we also determine the complexity of the plan compliance problem, that is, finding a plan that leads from an initial state to a desired goal state without reaching any undesired critical state. We consider all actions to be balanced, *i.e.*, their pre and post-conditions have the same number of facts. Under this assumption on actions, we show that the plan compliance problem is PSPACE-complete when all actions have only deterministic effects and is EXPTIME-complete when actions may have non-deterministic effects.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, F.4 Mathematical Logic and Formal Languages, J.3 Life and Medical Sciences

Keywords and phrases Multiset Rewrite Systems, Collaborative Systems, Applications of Rewrite Systems, Clinical Investigations, Maude

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.305

Category Regular Research Paper

1 Introduction

While carrying out a clinical investigation (CI)—that is, a set of procedures in medical research and drug development, to test a new drug or other intervention on human subjects, it is important that conclusive data is collected and that the health of the subjects participating in the CI is not compromised. In order to collect the most conclusive data, for instance, drug samples have to be taken and all the necessary tests have to be carried out in well defined periods of time. Moreover, since these experiments might compromise the health of subjects, CIs are rigorously regulated by policies elaborated by governmental agencies such as the Food and Drug Administration (FDA) [13]. These regulations require prompt action whenever a serious and unexpected problem with any subject is reported. In the current state of affairs, there is little to almost no automation in the management of CIs and therefore the process is prone to human error. As described in [23], there is plenty of room for the use of automated assistants to help reduce human mistakes from happening. For instance, a computer assistant can automatically generate plans that guide the clinical staff on how a CI has to be carried out. An assistant can also monitor the execution of a CI and signal alarms whenever a deviation to the specification is detected.

This paper proposes a rewriting framework that can be used to specify collaborative systems, such as CIs, and can be used as the foundations for building automated assistants. Our model is an extension of the systems used for modelling collaborative systems proposed in [18] with explicit time. An important feature of our model is that its specifications can be directly written and executed in Maude [6], a powerful tool based on rewrite logic [21]. By using its search mechanisms, Maude can be used to automatically generate plans from a specification. Moreover, one can rely on all the existing machinery and optimizations implemented in Maude. For instance, since Maude implements rewriting modulo axioms, the execution of systems with commutative and associative constructors, such as those building multisets, is greatly improved when using Maude.

A second feature of our framework is that its specifications can mention time explicitly. Time is often a key component used in policies specifying the rules and the requirements of a collaboration. For a correct collaboration and to achieve a common goal, participants should usually follow strict deadlines and should have quick reactions to some (unexpected) event. For instance, the paragraph 312.32 on Investigational New Drug Application (IND) safety [13] includes explicit time intervals that must be followed in case of any unexpected, serious or life-threatening adverse drug experience: (The emphasis in the text is ours.)

> “ (c) IND safety reports
> (1) Written reports –(i) The sponsor shall notify FDA and all participating investigators in a written IND safety report of: (A) Any adverse experience associated with the use of the drug that is both serious and unexpected; $[\cdots]$ Each notification shall be made as soon as possible and *in no event later than 15 calendar days* after the sponsor's initial receipt of the information $[\cdots]$
> (2) Telephone and facsimile transmission safety reports. The sponsor shall also notify FDA by telephone or by facsimile transmission of any unexpected fatal or life-threatening experience associated with the use of the drug as soon as possible but *in no event later than 7 calendar days* after the sponsor's initial receipt of the information.”

The clause above explicitly mentions two different time intervals. The first one specifies that a detailed safety report must be sent to the FDA within 15 days after a serious and unexpected event is detected, while the second specifies the obligation of notifying FDA of such an event within 7 days.

In order to accommodate explicit time, we attach to facts a *natural number* called *timestamp*. Timestamps can be used in different ways depending on the system being modeled. In the example above, the timestamp t of the fact $\mathit{Dose}(id)@t$ could denote that the subject with anonymous identification number id received a dose at time t. Alernatively, the timestamp, t_2, of the fact $\mathit{Deadline}@t_2$ could denote the time of when some activity should end. Moreover, we keep track of time by assuming a discrete global time, using the special fact $Time@t$, which denotes that the current time is t. The global time advances by replacing $Time@t$ by $Time@(t+1)$.

Agents change the state of the system by performing actions. In order to specify the type of time requirements illustrated above, a set of *time constraints* may be attached to actions This set acts as a guard of the action. A time constraint is a comparison involving exactly two timestamps, *e.g.*, $T_1 \leq T_2 + 7$ (see Eq. 1).

Besides allowing guards with time constraints, we also allow actions to have non-deterministic effects. In particular, actions are allowed to have a finite number of post-conditions specifying a finite number of possible resulting states. These actions are useful when specifying systems, such as CIs, whose actions may lead to different outcomes, but it is not certain beforehand which one of the outcomes will actually occur. For instance, when carrying out a blood test for the presence of some substance, it is not clear a priori what the test result will be. However, one can classify any result as either *positive* or *negative*. Depending on this result, one would need to take a different set of future actions. For example, if the blood test is positive, then one might not be suitable for participating as a subject in a particular CI, but may be suitable for other CIs. We classify actions that have more than one outcome as *branching actions*.

Finally, in collaborative systems agents collaborate in order to achieve a common goal, but they should also avoid *critical states* that, for example, violate policies. An example of a goal state for CIs would be to collect conclusive data without compromising the health of subjects, while a critical state would be a state that violates the FDA policies. In our model, critical, goal and initial states can also mention time explicitly by using time constraints.

This paper's contribution is twofold.

1. We determine the complexity of the *plan compliance* problem [18], that is, the problem of determining whether there is a plan where the collaboration achieves the common goal and in the process no critical state is reached. It has been shown that the plan compliance problem is undecidable in general [16]. However, we get decidability in the important case when all actions are *balanced*, *i.e.*, pre and post-conditions of actions have the same number of facts. Intuitively, this restriction bounds the memory of agents, as they can remember at any point only a bounded number of facts. Additionally, we assume that the facts created by an action, that is, the new facts that appear in its postcondition, can only have timestamps of the form $T + d$, where T is the current global time and d a natural number. Under these two assumptions on actions, we show that (1) the plan compliance problem is PSPACE-complete if no branching actions are allowed and (2) is EXPTIME-complete if branching actions are allowed.
2. We describe a Maude implementation of a small scenario of a clinical investigation visit specified in our rewriting model. We show how the search capabilities of Maude can be used for planning and compliance checking (run-time monitoring). Furthermore, we show how to improve execution performance by using two existing optimizations. The first optimization was proposed in [25], where it is shown how one can reduce search space due to interleavings by merging a collection of (small-step) rules that do not interfere with each other into a single (big-step) rule. The second optimization is to reduce the state space by declaring plan branch formation to be commutative as Maude works on equivalence classes modulo axioms. Our experiments demonstrate that it is possible to reduce in average the number of states to be traversed by a factor of 58 and search time by a factor of 118 when using such optimizations.

Regarding contribution **1** described above, even in the case of balanced actions, we have to deal with the problem that a plan can generate unbounded timestamps T. In particular, the state space is internally infinite since an *arbitrary* number of time advances can occur (as illustrated at the beginning of Section 4). In our previous work [15] we were able to solve a similar unboundedness problem caused by the presence of freshly created objects that are called *nonces* in protocol security literature. However, the solution proposed in [15] is not applicable to the problem of unboundedness of time. As a result, in this paper we have made special precautions in our choice of a novel equivalence relation among states based on the time differences of the timestamps of facts. This allows us to cover all plans of unbounded length caused by uncontrolled time advances, with providing our upper bounds for the timed collaborative systems (Theorem 6). We also show that our new technique introduced in this paper can be combined with the technique introduced in [15] to solve the unboundedness for both time and nonces in timed systems. In our experiments, we used this novel equivalence relation among states.

The paper is organized as follows. Section 2 introduces the formal model for timed collaborative systems called Timed Local State Transition Systems (*TLSTS*) as well as the plan compliance problem described above. (In [23], *TLSTS*es were only mentioned, but not formally introduced.) Section 3 expands the discussion in [23] on how to implement *TLSTS* specifications in Maude. Section 4 introduces an equivalence relation between states of the system that allows us to handle the unboundedness of time with finite space. The machinery introduced in this section is used in Section 5 to demonstrate the decidability of the plan compliance problem, and in our experiments in Section 6. Section 5 contains the complexity results mentioned above. Section 6 explains the scenario and the experimental results obtained. Finally in Sections 7 and 8 we discuss related and future work.

2 Basic Definitions

At the lowest level, we have a first-order alphabet Σ that consists of a set of predicate symbols $P_1, P_2, \ldots$, function symbols $f_1, f_2, \ldots$, constant symbols $c_1, c_2, \ldots$, and variable symbols $x_1, x_2, \ldots$ all with specific sorts (or types). The multi-sorted terms over the alphabet are expressions formed by applying functions to arguments of the correct sort. Since terms may contain variables, all variables must have associated sorts. A fact is an atomic predicate over multi-sorted terms.

In order to accommodate time in our model, we associate to each fact a timestamp. *Timestamped facts* are of the form $P(t_1, \ldots, t_n)@t$, where t is the timestamp of the fact $P(t_1, \ldots, t_n)$. Among the set of predicates, we distinguish the zero arity predicate *Time*, which intuitively denotes the current global time of the system. For instance, the fact *Time*@2 denotes that the global time is 2. Here, we assume that timestamps are natural numbers. The intuitive meaning of a timestamp may depend on the system one is modeling. For instance, in our clinical investigations example, the timestamp associated to a fact could denote the time when a problem with a subject has been detected.

The *size of a fact*, P, denoted by $|P|$, is the total number of symbols it contains. We count one for each constant, variable, predicate, and function symbols, *e.g.*, $|P(x, c, x)| = 4$, and $|P(f(x))| = 3$. For our complexity results, we assume an upper bound on the size of facts, as in [12, 18, 15]. This means that for all facts, $P(t_1, \ldots, t_n)@t$, the arity of symbols, n, and the depth of terms, $t_1, \ldots, t_n$, are bounded. However, we make no assumptions on the depth of timestamps, t, that is, the size of timestamps may be unbounded.

A *state*, or *configuration* of the system is a finite multiset, $Q_1@t_1, \ldots, Q_n@t_n$, of *grounded* timestamped facts, *i.e.*, timestamped facts not containing variables. Configurations are assumed to contain exactly one occurrence of the predicate *Time*. We use W, X to denote the multiset resulting from the multiset union of W and X. For instance, the configuration

$$\{Time@5, Blood(id_1, scheduled)@7, Dose(id_1)@5, Status(id_1, normal)@5\}$$

denotes that that current time is 5, that the blood test for subject identified by id_1 should be taken on time 7, that the same subject took a dose of the drug at time 5, and his status is *normal*, *i.e.*, no problem has been detected.

Following [18], we assume that the global configuration is partitioned into different local configurations each of which is accessible only to one agent. There is also a public configuration, which is accessible to all agents. This separation of the global configuration is done by partitioning the set of predicate symbols in the alphabet and it will be usually clear from the context. The time predicate *Time* is assumed to be public. For instance, in the configuration above all facts, except *Time*, belong to the health institution monitoring the subject id_1.

Time constraints. The time requirements of a system are specified by using *time constraints*. Time constraints are arithmetic comparisons involving exactly two timestamps:

$$T_1 = T_2 \pm d,\ T_1 > T_2 \pm d,\ \text{ or } T_1 \geq T_2 \pm d, \tag{1}$$

where d is a natural number and T_1 and T_2 are time variables, which may be instantiated by the timestamps of any fact including the global time.

A concrete motivation for time constraints to be relative is that, as in physics, the rules of a collaboration are also not affected by time shifts. If we shift the timestamps of all facts by the same value, the same rules and conditions valid with respect to the original state are also valid with respect to the resulting state. If time constraints were not relative, however, then one would not be able to establish this important invariant. Indeed, as we show in the companion technical report [22], the reachability problem is undecidable for systems with non-relative time constraints.

Branching Actions and Branching Plans. Actions work as multiset rewrite rules. As in [18, 15] we assume that each agent has a finite set of actions. However, we extend actions in two different ways by adding guards to actions and by allowing actions to have non-deterministic effects.

In their most general form, actions have the following shape:

$$W \mid \Upsilon \ \longrightarrow_A \ [\exists \vec{x_1}.W_1] \oplus \cdots \oplus [\exists \vec{x_n}.W_n] \tag{2}$$

The subscript A is the name of the agent that owns this action. W is the pre-condition of this rule, while $W_1, \ldots, W_n$ are its post-conditions. All facts in $W, W_1, \ldots, W_n$ are public and/or belong to the agent A. Υ is the guard of the action consisting of a finitely many *time constraints*. The existentially quantified variables specify the creation of fresh values, also known as nonces in protocol security literature. If $n > 1$, then we classify this action as *branching*, otherwise when $n = 1$ we classify this action as *non-branching*.

A *branching plan* is a tree whose nodes are configurations and whose edges are labeled with a pair consisting of an action and a number, $\langle \alpha, i \rangle$. A plan is constructed by applying an action to one of its leaves. Formally, when a branching action α of the form shown in Eq. 2 is applied to a leaf of a plan labeled with W_I, the corresponding branch of the plan is extended by adding n leaves. The configuration labeling the i^{th} leaf is obtained by replacing α's pre-condition W in W_I by the post-condition W_i of α. The edge connecting W_I with i^{th} new leaf is labeled with $\langle \alpha, i \rangle$. In the process fresh values are created, which replace the existentially quantified variables, $\vec{x_i}$.[1]

For example, let $\{Time@6, P(t_1)@1, Q(t_2)@4\}$ be a configuration appearing in the leaf of a plan $\mathcal{P}$. Then the following branching action is applicable:

$Time@T, Q(Y)@T_1 \mid \{T > T_1 + 1\} \longrightarrow_A [\exists x.Time@T, R(Y,x)@T] \oplus [Time@T, S(Y)@T]$

and it extends the plan $\mathcal{P}$ creating the following two leaves $\{Time@6, P(t_1)@1, R(t_2, n)@6\}$ and $\{Time@6, P(t_1)@1, S(t_2)@6\}$, where n is a fresh value.

We will assume that the only action that can change the global time is the following action belonging to the special agent *clock*:

$$Time@T \mid \{\} \rightarrow_{clock} Time@(T+1). \tag{3}$$

The action above does not have any constraints, which is specified by the empty set $\{\}$. It is the only action of the agent *clock*.

For the actions belonging to the remaining agents, we will further impose the following two conditions on actions depicted in Eq. 2. Firstly, the global time $Time@T$ appears in the pre-condition, W, and in each of the post-conditions $W_1, \ldots, W_n$ exactly once. Secondly, if $Time@T$ is in the pre-condition W, then all facts created in the post-conditions $W_1, \ldots, W_n$ are of the form $P@(T+d)$, where d is a natural number, possibly zero. That is, all the created facts have timestamps greater or equal to the global time. Notice that in this type of actions the timestamp of *Time* does not change, that is, actions are *instantaneous*. For instance, the following action is not allowed:

$Time@T, R@T_1, P@T_2 \mid T_1 < T \longrightarrow_A Time@T, R@T_1, S@T_1$

because the timestamp of the created fact $S@T_1$ is not of the form $(T+d)$. That is, actions are only allowed to create facts whose timestamps are in the present or in the future.

► **Definition 1.** A *timed local state transition system* (*TLSTS*) $\mathcal{T}$ is a tuple $\langle \Sigma, I, R_{\mathcal{T}} \rangle$, where Σ is the alphabet of the language, I is a set of agents, such that $clock \in I$, and $R_{\mathcal{T}}$ is a finite set of actions owned by the agents in I of the two forms described above.

We classify an action as *balanced* if its post-conditions, W_i, and the pre-condition, W, have the same number of facts (see Eq. 2). As discussed in [18], if all actions in a system are balanced, then the size of all configurations in a plan remains the same as in the initial configuration. Since we assume facts to have a bounded size, the use of balanced actions imposes a bound on the storage capacity of the agents in the system.

Timed Initial, Goal and Critical Configurations. In a collaboration, agents interact in order to achieve some common goal. However, since they do not trust each other completely, they also want to avoid some critical situations. Often these goals and critical situations mention time explicitly. For instance, in clinical investigations example discussed in the Introduction, the participants want to collect conclusive data without violating regulations. The sponsor should send a safety report to the FDA whenever a serious and unexpected problem is detected within 15 days. Otherwise, the sponsor can be severely penalized.

In order to formalize such aspects of a collaboration, we extend the notion of initial, goal and critical configurations proposed in [18] by attaching a set of time constraints. In particular, timed initial, goal and critical configurations have the following form:

$\{Q_1@T_1, Q_2@T_2, \ldots, Q_n@T_n\} \mid \Upsilon$

[1] Fresh values are often used in administrative processes, such as when a transaction number is issued. In particular, the transaction number has to be fresh. For a more detailed account for nonce see [15].

where Υ is a finite set of time constraints as shown in Eq. 1 and whose variables are in $T_1, T_2, \ldots, T_n$.

For instance, in the clinical investigations example, a possible goal configuration is when the data of a subject is collected in specified intervals for some number of times. The following goal configuration specifies that the goal is to collect the data of a subject 25 times in intervals of 28 days, but with a tolerance of 5 days: $\{Time@T, Data(Id, 1)@T_1, \ldots, Data(Id, 25)@T_{25}\}$ with the time constraints $T_i + 23 \leq T_{i+1} \leq T_i + 33$ and that $T > T_i$, for $1 \leq i \leq 25$. Formally, any instantiation of the variables $T_1, \ldots, T_{25}$ that satisfy the set of constraints above is considered a goal configuration.

Similarly, a configuration is critical for the participants of a clinical investigation when a problem is detected at time T_1, but the written report is not sent to the FDA on time, *i.e.*, within 15 days after the problem is detected: $\{Detect(Id)@T_1, Report(Id)@T_2\} \mid \{T_2 > T_1 + 15\}$.

Adding time constraints to configurations is not a restriction of the model. Quite the contrary, we provide much more flexibility within our formalism by either not constraining configurations at all or by providing a kind of temporal interference to specify initial/goal/critical configurations, in addition to the timestamps given separately.

For simplicity, we often omit the word "timed" in initial/goal/critical configurations regardless of time constraints being attached or not.

Planning Problem. In [16], three compliance problems were introduced in the setting without explicit time or branching (actions with non-deterministic effects). We now restate one of these problems, called plan compliance problem in our setting with explicit time and branching. The remaining problems are dealt in detail in the technical report [22].

Given an initial configuration W and a finite set of goal and critical configurations, we call a *branching plan* $\mathcal{P}$ *compliant* if it does not contain any critical configurations and moreover if all branches of $\mathcal{P}$ lead from configuration W to a goal configuration.

(*Plan compliance problem*) Given a timed local state transition system $\mathcal{T}$, an initial configuration W consisting of grounded timestamped facts and a finite, possibly empty, set of time constraints, a finite set of goal and a finite set of critical configurations, is there a compliant plan?

3 Implementing a *TLSTS* in Maude

The general-purpose computational tool Maude [6] provides all the machinery necessary to implement directly *TLSTS* specifications. As Maude is based on rewriting, the Maude code looks similar to the specification itself. We now illustrate by using examples of how the encoding works.

Configurations. We start by specifying the signature of a TLSTS, *i.e.*, the set of constants and predicate symbols. For instance, the code below specifies that the zero arity fact `time` is of sort (or type) `Fact` and that `blood` is a binary fact whose argument is of sort `Id` and `Result`.

```
op time :  -> Fact .     op blood :  Id Result -> Fact .
```

Other predicates of the sort `Fact` can be specified in a similar fashion.

We specify as follows the operator `@` which attaches a natural number to facts and are used to specify timestamped facts which are of sort `TFact`.

```
op _@_ :  Fact Nat -> TFact .
```

To encode configurations, we first specify that timestamped facts is a subsort of configuration, denoted by the symbol `<`, that the empty set is a configuration, specified by the operator `none`, and that the juxtaposition of two configurations is also a configuration.

```
subsort TFact < Conf .
op none :  -> Conf .     op __ : Conf Conf -> Conf [assoc comm id: none] .
```

The last statement also specifies that configurations are multisets by attaching the keywords `assoc` and `comm`, which specify that the operator constructing configurations is both associative and commutative. Hence, when Maude checks whether an action (specified below) is applicable, Maude will consider all possible permutations of elements until it finds a match which satisfies the action's pre-condition as well as its guard. Finally, the keyword `id: none` specifies that the constructor `none`, specifying the empty set, is the identity of an operator. For instance, it is used to identify the configurations `none (time@2) none (blood(id1,positive)@3)` and `(time@2) (blood(id1,positive)@3))`.

Timed Critical and Timed Goal Configurations. Timed critical and timed goal configurations are specified as *equational theories.* For instance, the following equational theory specifies in Maude the critical configuration when the FDA is not notified 7 days after a serious and unexpected problem is detected. Here `Num` is the fresh value, *e.g.*, a number, uniquely identifying a serious and unexpected event with subject identified by `Id`.

```
ceq critical((C:Conf)(time@T)(detected(Id,Num)@T1)(fda(Id,no,Num)@T2))
       = true   if T > T1 + 7
```

Maude automatically replaces `critical(C)` by the boolean `true` if the configuration `C` satisfies the condition specified by the equation above. Timed goal configurations are also specified as equational theories in a similar way, only that we use the predicate `goal`, instead of `critical` to specify goal configurations.

Branching Actions and Searching for Compliant Plans. Whereas critical and goal configurations are specified by using equational theories, actions are specified as rewrite rules in Maude. To accommodate branching actions, we use three new operators `noPlan`, denoting when a branching plan has no leaves, brackets used to mark a leaf of a plan, and `+` used to construct the list of leaves of a branching plan. The leaves of a branching plan belong to the sort `Plan`.

```
op noPlan :  -> Plan .
op {_} :  Conf -> Plan .   op _+_  :  Plan Plan-> Plan [assoc id:noPlan] .
```

The operator `+` is also used to specify the different outcomes of an action. For instance, the following conditional rule specifies that there are two possible outcomes when a blood test `scheduled` at time `T1` is carried out, namely, the blood test is `positive` or `negative`. Moreover, the boolean conditions specifies that test can only be carried out at the same day when it was scheduled and if none of its outcomes is a critical configuration.

```
crl[blood]: {(C:Conf)(time@T)(blood(Id,scheduled)@T1)} =>
                 {(C:Conf)(time@T)(blood(Id,positive)@T)} +
                 {(C:Conf) (time@T) (blood(Id,negative)@T)}
  if T1 = T ∧ not (critical((C:Conf)(time@T)(blood(Id,positive)@T))) ∧
    not (critical((C:Conf)(time@T)(blood(Id,negative)@T)))
```

Formally, when this rule is applied then two different leaves are created, one for each possible result. The remaining facts appearing in the configuration `C` are left untouched.

Notice that the definition of the `_+_` operator does not specify it to be commutative. However, regarding the compliance problem that we are interested on (described in Section 2), changing the order of the branches of a plan preserves its compliance as the resulting plan does not reach any critical configuration and each of its leaves are goal configurations. Thus, we can safely change the definition of `_+_` to also be commutative. As we demonstrate in Section 6, this change reduces in average by a factor of 8 the number of possible states.

As in the rule above, we allow a rule to be applied only if *all* its outcomes are *not* critical configurations. For instance, the action that advances time (Eq. 3) is specified in Maude with an extra condition allowing the time to be incremented only if the resulting configuration is not critical:

```
    crl[time]: {(C:Conf)(time@T)} => {(C:Conf)(time@(T+1))}
      if not (critical((C:Conf)(time@(T+1))))
```

This means that it is not possible to reach a critical configuration when using the rules as encoded above. Therefore, in order to search for a compliant plan, one does not need to care whether a critical configuration is reached, as this is not possible, but only check whether there is a plan from an initial configuration to a goal configuration obtained by using the actions as mentioned above. Maude can automatically perform this search by using a command of the following form:

```
 search in MODULE_NAME : I =>+ P:Plan such that goals(P:Plan) = true .
```

where `I` is the initial configuration, `MODULE_NAME` is the name of the Maude module containing all the rules of the *TLSTS*, and finally `goals` is an equational theory that returns true when given $\{C_1\}$ + $\cdots$ + $\{C_n\}$ of type `Plan` only if `goal`(C_i) evaluates to true for all $1 \leq i \leq n$.

It is often possible to demonstrate the non-interference of two actions, α and β, syntactically. For instance, if there is no intersection between the facts modified by α and β, these actions do not interfere between each other as they mention different parts of a configuration. The following action specifying a vital sign test does not interfere with the action above specifying a blood test:

```
    crl[vital]: {(C:Conf)(time @ T)(vital(I,ID,false)@T1))} =>
                         {(C:Conf)(time @ T)(vital(I,ID,true)@T)}
      if T1 = T ∧ not (critical((C:Conf)(time@T)(vital(I,ID,true)@T))
```

This means that a compliant plan containing a sequence of actions $\alpha; \beta$ can be replaced by another compliant plan where the order is inverted $\beta; \alpha$. In our example scenario, such interleavings increase the number os states Maude must explore by a factor of 23. A better approach would be to merge these actions into a (big-step) action. For example, the big-step action obtained from the two actions above would specify the actions of performing the vital signs and blood test at the same time. For instance, one of its post-conditions specifies when the blood test is positive:

```
  {(C:Conf)(time @ T)(vital(I,ID,true)@T)(blood(Id,positive)@T)}.
```

Finally, besides searching for plans, the same theory can also be used for monitoring CI executions. For instance, by using the equational theory specifying critical configurations, one can detect when a deviation has occurred and send alarms to the responsible agents. After a CI has been carried out, one could also use the actual plan carried out to study how CIs have been executed.

4 Dealing with the Unboundedness of Time

Comparing our *timed* collaborative models introduced here with the results on the *untimed* collaborative systems in our previous work, we meet with a number of the crucial difficulties. In the case of planning problems for the untimed systems with balanced actions, we are dealing with a *finite* (though huge) state space. Here the state space is *internally infinite*, since an arbitrary number of time advances is allowed in principle. For a straightforward example, consider a plan where time is eagerly advanced. That is, consider a plan with a single branch where time advances constantly:

$$Time@0, W \longrightarrow_{clock} Time@1, W \longrightarrow_{clock} Time@2, W \longrightarrow_{clock} \cdots$$

Since there are no bounds on the length nor depth of plans, the final value of the global time cannot be bounded in advance.

This section describes how to overcome the problem above by proposing an equivalence relation between configurations. The key idea is that since time constraints are relative, that is, they involve the difference of two timestamps, we do not need to keep track of the values for timestamps separately, in order to determine whether our time constraints are satisfied or not.

Truncated time differences. In particular, we will store the time differences among the facts, but truncated by an upper bound. Formally, assume D_{max} be an upper bound on the numbers appearing explicitly in a given planning problem with the model $\mathcal{T}$ - that is, the numbers in the actions and time constraints in $\mathcal{T}$, and in the initial, goal and critical configurations, for instance, the d in Eq. 1. Then the *truncated time difference* of two timed facts $P@T_1$ and $Q@T_2$ with $T_1 \leq T_2$, denoted by $\delta_{P,Q}$, is defined as follows:

$$\delta_{P,Q} = \begin{cases} T_2 - T_1, \text{ provided } T_2 - T_1 \leq D_{max} \\ \infty, \text{ otherwise} \end{cases}$$

Intuitively, we can truncate time differences without sacrificing soundness nor completeness because time constraints are relative as shown in Eq. 1. Hence, if the time difference of two facts is greater than the upper bound D_{max}, then it does not really matter how much greater it is, but just that it is greater. For instance, consider the time constraint $t_1 \geq t_2 + d$ involving the timestamps of the facts $P@t_1$ and $Q@t_2$. If $\delta_{Q,P} = \infty$, this time constraint is necessarily satisfied.

Equivalence between configurations. We use the notion of truncated time differences introduced above to formalize the following equivalence relation among configurations.

▶ **Definition 2.** Given a planning problem with the *TLSTS* $\mathcal{T}$, let D_{max} be an upper bound on the the numeric values appearing in $\mathcal{T}$ and in the initial, goal and critical configurations. Let

$$\mathcal{S} = Q_1@T_1, Q_2@T_2, \ldots, Q_n@T_n \quad \text{and} \quad \widetilde{\mathcal{S}} = Q_1@\widetilde{T}_1, Q_2@\widetilde{T}_2, \ldots, Q_n@\widetilde{T}_n$$

be two configurations written in canonical way where the two sequences of timestamps $T_1, \ldots, T_n$ and $\widetilde{T}_1, \ldots, \widetilde{T}_n$ are non-decreasing. (For the case of equal timestamps, we sort the facts in alphabetical order, if necessary.) Then $\mathcal{S}$ and $\widetilde{\mathcal{S}}$ are equivalent if for any $1 \leq i < n$ either of the following holds: $T_{i+1} - T_i = \widetilde{T}_{i+1} - \widetilde{T}_i \leq D_{max}$ or both $T_{i+1} - T_i > D_{max}$ and $\widetilde{T}_{i+1} - \widetilde{T}_i > D_{max}$.

To illustrate the equivalence above, assume that $D_{max} = 3$ and consider the following two configurations: $\{R@3, P@4, \mathit{Time}@11, Q@12, S@14\}$ and $\{R@0, P@1, \mathit{Time}@6, Q@7, S@9\}$. According to the definition above, these configurations are equivalent since their truncated time differences are the same. This can be observed by the following *canonical* representation, called *δ-representation.* A δ-representation is constructed from a given configuration by sorting its facts according to their timestamps and sorting facts in alphabetical order as tie-breaker. Then we compute the time difference among two consequent facts, $\delta_{Q_i,Q_{i+1}}$. For instance, both configurations above have the following δ-representation: $\langle R, 1, P, \infty, \mathit{Time}, 1, Q, 2, S\rangle$

Here a value appearing between two facts, Q_i and Q_{i+1}, is the truncated time difference of the corresponding facts, $\delta_{Q_i,Q_{i+1}}$, *e.g.*, $\delta_{R,P} = 1$ and $\delta_{P,\text{Time}} = \infty$. It is also easy to see that from the tuple above, one can compute the remaining truncated time differences. For instance, $\delta_{\text{Time},S} = 3$, since $1 + 2 = 3$, while $\delta_{R,Q} = \infty$, since $1 + \infty + 1 = \infty$.

We now formalize the intuition described above that using time differences that are truncated by an upper bound is enough to determine whether a time constraint is satisfied or not.

▶ **Lemma 3.** *Let S and $\widetilde{S}$ be two equivalent configurations from Definition 2.*
$\mathcal{S} = Q_1@T_1, Q_2@T_2, \ldots, Q_n@T_n$ *and* $\widetilde{\mathcal{S}} = Q_1@\widetilde{T}_1, Q_2@\widetilde{T}_2, \ldots, Q_n@\widetilde{T}_n$.
Then the following holds for all i and j such that $i > j$, and for all $a \leq D_{max}$:

$T_i - T_j = a$ *if and only if* $\widetilde{T}_i - \widetilde{T}_j = a$
$T_i - T_j < a$ *if and only if* $\widetilde{T}_i - \widetilde{T}_j < a$
$T_i - T_j > a$ *if and only if* $\widetilde{T}_i - \widetilde{T}_j > a$

Proof The only interesting case is the last one, which can be proved by using the fact that $a \leq D_{max}$ and that S and $\widetilde{S}$ are equivalent. Hence, $T_i - T_j > D_{max} > a$ is true if and only if $\widetilde{T}_i - \widetilde{T}_j > D_{max} > a$ is true, and $D_{max} \geq T_i - T_j > a$ is true if and only if $D_{max} \geq \widetilde{T}_i - \widetilde{T}_j > a$, since $T_i - T_j = \widetilde{T}_i - \widetilde{T}_j$. ◀

Handling time advances and action applications. Our next task is to show that our equivalence relation using truncated time differences is well-defined with respect to actions. That is, we show that actions preserve the equivalence among configurations. This will allow us to represent plans using δ-representations only.

However, in order to prove such a result, we need yet another assumption on configurations in order to faithfully handle time advances. The problem lies with the *future facts*, that is, those facts whose timestamps are greater than the global time. If there is a future fact P such that $\delta_{\text{Time},P} = \infty$, then it is not the case that equivalence is preserved when we advance time. For example, consider the following two configurations equivalent under the upper bound $D_{max} = 3$:

$\mathcal{S}_1 = \{Time@0, P@5\}$ and $\mathcal{S}_2 = \{Time@0, P@4\}$.

If we advance time on both configurations, then the resulting configurations, $\mathcal{S}_1'$ and $\mathcal{S}_2'$, are not equivalent. This is because the truncated time difference $\delta_{\text{Time},P}$ is still ∞ in $\mathcal{S}_1'$, while it changes to 3 in $\mathcal{S}_2'$. Notice that the same problem does not occur neither with present nor past facts, *i.e.*, those facts whose timestamps are less or equal to the global time.

▶ **Definition 4.** Given an upper bound D_{max} in a planning problem (as per Definition 2), a configuration $\mathcal{S}$ is called *future bounded* if for any future fact P in $\mathcal{S}$, the time difference $\delta_{\text{Time},P} \leq D_{max}$.

Recall from Section 2 that there are two types of actions, namely, the action that advances time and instantaneous actions belonging to agents. Moreover, recall that the latter actions are restricted in such a way that all created facts have timestamps of the form $T + d$, where T is the global time. This restriction allows us to show that actions preserve the future boundedness of configurations as states the following result.

▶ **Lemma 5.** *Let $\mathcal{T}$ be a TLSTS and $\mathcal{S}$ be a future bounded configuration. Let $\mathcal{S}'$ be the configuration obtained from $\mathcal{S}$ by applying an arbitrary action in $\mathcal{T}$. Then $\mathcal{S}'$ is also future bounded.*

As per Definition 2 the initial configuration in a planning problem is future bounded, which from the lemma above implies that all configurations in a plan are also future bounded. Notice that even if we relax the assumption that the initial configuration is future bounded, we can make it be future bounded by setting the value of D_{max} to be the greatest timestamp in the initial configuration, *i.e.*, D_{max} is the upper bound on the values of the given *TLSTS* and in the initial, goal, and critical configurations. The important result is the lemma above that states that future boundedness is preserved by actions.

We are now ready to show the main result of this section.

Table 1 Summary of the complexity results for the plan compliance problem. We mark the new results appearing here with a $\star$. For the undecidability result, [16] shows that the plan compliance problem is undecidable even when actions are not allowed to create fresh values.

Balanced Actions	Non-Branching	PSPACE-complete*
	Possibly Branching	EXPTIME-complete*
Not Necessarily Balanced Actions		Undecidable [16]

▶ **Theorem 6.** *For any given planning problem the equivalence relation between configurations given by Definition 2 is well-defined with respect to the actions of the system (including time advances) and goal and critical configurations. Any plan starting from the given initial configuration can be conceived as a plan over δ-representations.*

Proof (Sketch) Let $\mathcal{S}$ and $\widetilde{\mathcal{S}}$ be two equivalent configurations. Assume that $\mathcal{S}$ is transformed in $\mathcal{S}'$ by means of an action α. By Lemma 3 the configuration $\widetilde{\mathcal{S}}$ also complies with the time constraints required in α, and hence the action α will transform $\widetilde{\mathcal{S}}$ into some $\widetilde{\mathcal{S}}'$. It remains to show that $\widetilde{\mathcal{S}}'$ is equivalent to $\mathcal{S}'$.

We consider our two types of actions. Let the time advance transform $\mathcal{S}$ into $\mathcal{S}'$, and $\widetilde{\mathcal{S}}$ into $\widetilde{\mathcal{S}}'$. From Lemma 5, we have that both $\mathcal{S}'$ and $\widetilde{\mathcal{S}}'$ are future bounded and therefore $\mathcal{S}'$ and $\widetilde{\mathcal{S}}'$ are trivially equivalent. For the second type of actions, namely the instantaneous actions belonging to agents, the reasoning is similar. Each created fact in the configuration $\mathcal{S}'$ and $\widetilde{\mathcal{S}}'$ will be of the form $P@(T+d)$ and $P@(\widetilde{T}+d)$, where T and $\widetilde{T}$ are the global time in $\mathcal{S}$ and $\widetilde{\mathcal{S}}$, respectively. Therefore each created fact has the same difference d to the global time, which implies that these created facts have the same truncated time differences to the remaining facts. Hence $\mathcal{S}'$ and $\widetilde{\mathcal{S}}'$ are equivalent.

Finally, also from Lemma 3, $\mathcal{S}$ is a goal (respectively, critical) configuration if and only if $\widetilde{\mathcal{S}}$ is a goal (respectively, critical) configuration. ◀

The theorem above establishes that using δ-representations for writing plans is well defined, but it does not establish a bound on the number of δ-representations. To achieve this, we need the further assumption that all actions are balanced. Recall that balanced actions are actions that have the same number of facts in their pre and post-conditions. By using balanced actions, the number of facts in any configuration of a plan is the same as the number of facts in the plan's initial configuration. Hence, we can also establish that there is a finite number of δ-representations. Later in the Section 5 we provide more precise bounds.

▶ **Corollary 7.** *In the case of a model $\mathcal{T}$ with balanced actions, we can deal the planning problem with the* finite *space of representatives of the form*
$(Q_1, \delta_{12}, Q_2, \delta_{23}, Q_3, \ldots, Q_i, \delta_{i,i+1}, Q_{i+1}, \ldots, Q_m, \delta_{m,m+1}, Q_{m+1})$.
The size of each of the representatives is polynomial with respect to m, k, and $\log_2 D_{max}$, where m is the number of facts in the initial configuration, k is the upper bound on the size of facts, and D_{max} is the upper bound on the numeric values appearing in $\mathcal{T}$ and in the initial, goal and critical configurations.

5 Complexity Results

This section enters into the details of the complexity of the plan compliance problem described at the end of Section 2. Throughout this section, we assume that all actions are balanced, *i.e.*, actions have the same number of facts in their pre and post-conditions.

Our main results are summarized in Table 1.

Plan Compliance with Non-Branching Actions only. We consider the plan compliance problem when actions are non-branching and balanced and when the size of facts is bounded. We show that this problem is PSPACE-complete.

PSPACE-hardness: It was shown in [15] that one can faithfully encode a Turing machine with tape of size n using systems with balanced actions. The same idea works in our setting with time. It is easy to modify the encoding in [15]. Timestamps do not play any important role in such encoding.

PSPACE upper bound: It is more interesting to show that the plan compliance problem is in PSPACE when the size of facts is bounded and actions are non-branching and balanced. In particular, we will now use all the machinery introduced in Section 4 by using δ-representations of configurations to search for compliant plans.

In order to determine the existence of a compliant plan, it is enough to consider plans that never reach configurations with the same δ-configuration twice. If a plan reaches to a configuration whose δ-representation is the same as a previously reached configuration, there is a cycle of actions which could have been avoided. The following lemma imposes an upper bound on the number of different δ-representations in a plan given an initial finite alphabet. Such an upper bound provides us with the maximal length of a plan one needs to consider.

▶ **Lemma 8.** *Given a TLSTS $\mathcal{T}$ under a finite alphabet Σ, an upper bound on the size of facts, k, and an upper bound, D_{max}, on the numeric values appearing in the planning problem, namely, in $\mathcal{T}$ and in the initial, goal and critical configurations, then the number of different δ-representations, denoted by $L_T(m, k, D_{max})$, with m facts (counting repetitions) is such that $L_T(m, k, D_{max}) \leq (D_{max} + 2)^{(m-1)} J^m (D + 2mk)^{mk}$, where J and D are, respectively, the number of predicate and the number of constant and function symbols in the initial alphabet Σ.*

Proof Let $\langle Q_1, \delta_{Q_1,Q_2}, Q_2, \ldots, Q_{m-1}, \delta_{Q_{m-1},Q_m}, Q_m \rangle$ be a δ-representation with m facts. There are m slots for predicate names and at most mk slots for constants and function symbols. Constants can be either constants in the initial alphabet Σ or names for fresh values (nonces). Following [15], we need to consider only $2mk$ names for fresh values (nonces). Finally, only time differences up to D_{max} have to be considered together with the symbol ∞ and there are $m - 1$ slots for time differences in a δ-representation. ◀

Intuitively, our upper bound algorithm keeps track of the length of the plan it is constructing and if its length exceeds $L_T(m, k, D_{max})$, then it knows that it has reached the same δ-representation twice. This is possible in PSPACE since the number above, when stored in binary, occupies only polynomial space with respect to its parameters. For the result below, we assume that it is possible to check in polynomial space when a configuration is critical, when it is a goal configuration, and when an action is applicable in a configuration.

▶ **Theorem 9.** *Let $\mathcal{T}$ be a model with balanced non-branching actions. Then the plan compliance problem is in PSPACE with respect to m, k, and $\log_2 D_{max}$, where m is the number of facts in the initial configuration, k is the upper bound on the size of facts, and D_{max} is the upper bound on the numeric values appearing in the model $\mathcal{T}$, and in the initial, goal and critical configurations.*

Plan Compliance with possibly Branching Actions. We now consider the plan compliance problem when actions may also be branching. In particular, we show that when actions are balanced then the plan compliance problem is EXPTIME-complete with respect to the number of facts, m, in the initial configuration, the upper bound, k, on the size of facts, the upper bound, D_{max}, on the numbers explicitly appearing in the planning problem, and the upper bound, p, on the number of post-conditions of an action.

EXPTIME-hardness: The lower bound for the plan compliance problem can be inferred from a similar lower bound described in [19]. It was shown that one can encode alternating Turing machines [5] by using propositional actions that are balanced and branching. Time does not play an important role for that encoding.

EXPTIME upper bound: Our upper bound algorithm uses an alternating Turing machine. In particular, we show that the plan compliance problem is in alternating-PSPACE (APSPACE) with respect to the number of facts, m, in the initial configuration, the upper bound, D_{max}, on the numbers appearing explicitly in the planning problem, and the upper bound, p, on the number of post-conditions of any action. That is, an alternating Turing machine can solve the plan compliance problem using polynomial space. From the equivalence between APSPACE and EXPTIME shown in [5], we can infer that the plan compliance problem is in EXPTIME with respect the same parameters.

▶ **Theorem 10.** *Let $\mathcal{T}$ be a model with balanced actions. Then the plan compliance problem is in EXPTIME with respect to m, k, and $\log_2 D_{max}$, and p, where m is the number of facts in the initial configuration, k is the upper bound on the size of facts, and D_{max} is the upper bound on the numeric values appearing in the model $\mathcal{T}$, and in the initial, goal and critical configurations, and p is the upper bound on the number of post-conditions of actions in $\mathcal{T}$.*

6 Scenario Implemented and Experimental Results Summary

We implemented a small scenario simulating a visit of a subject in a clinical investigation. In this scenario, a subject has to undergo three tests, namely, *vital signs*, *hematology*, and *urine tests* and in some cases a further *nephrology test.* The first three tests have to be performed at the same day of the subject's visit. While the vital signs and hematology tests have a single outcome, where the data is collected, the results of the urine test may be classified in three levels: normal, high, or very high (typically over three times the normal upper bound). That is, the urine test has three outcomes according to the urine test result. If the result is very high, the urine test must be repeated *within five days*, in order to be sure that the first result is not an isolated result. Moreover, if the result of the second urine test is either high or very high, then an extra nephrology test must be performed on the same day as the second urine test. The visit is over when all necessary tests have been carried out.

As described in Section 3, tests are specified as a rewrite theory specifying an action, while the time conditions in the scenario are specified using the equational theory for critical configurations. In particular, the action for urine test has three outcomes, one for each possible result of the test. We have also implemented the machinery described in Section 4. For this example, it is enough to compute the canonical form whenever time advances.

For our experiments using Maude, we considered the following two optimizations. Since the order in which the leaves of a plan appear do not really matter, we can specify the `+` to be also commutative by adding the attribute `comm` to its definition. Since Maude implements rewriting modulo axioms, this reduces both the state space and the number of solutions. The second optimization, on the other hand, follows the lines described in [25] and involves avoiding interleavings of actions by merging (small-step) actions into larger (big-step) actions. However, in order to be sound and complete, such a merging of actions can only involve actions that are mutually independent. For instance, the order in which one performs the vital signs, the hematology and the first urine test is not important. Hence, instead of specifying each test as a different action, we can execute all three tests as a single action. Moreover, since the urine test has three possible outcomes, while the other test have only one outcome, the resulting (big-step) action will also have three possible outcomes.

Table 2 Summary of our experimental results with different optimizations, *e.g.*, big-step rules and commutative +. An entry of the form $n\,/\,t\,/\,s$ denotes that the search space had a total of n states and it took Maude t seconds to traverse all states finding s solutions. DNF denotes that Maude did not terminate after 40 minutes, C denotes Commutative and NC denotes Non-Commutative.

D_{max}		0	2	4	6	8
Small	NC	63k / 91 / 6	166k / 263 / 364	373k / 603 / 4k	755k / 1651 / 19k	DNF
Step	C	43k / 71 / 6	83k / 119 / 56	141k / 188 / 252	222k / 340 / 792	332k / 640 / 2k
Big	NC	3k / 3 / 6	14k / 14 / 364	51k / 67 / 4k	140k / 220 / 19k	329k / 508 / 66k
Step	C	1k / 1 / 6	3k / 2 / 56	6k / 5 / 252	13k / 13 / 792	23k / 26 / 2k

Table 2 summarizes our main experimental results for the scenario described above when using different parameters D_{max} with the upper-bound of numbers appearing anywhere in the theory (see Section 4) as well as the two optimizations described above. We performed these experiments on an Ubuntu machine (Kernel 2.6.32-37) with 3.7 Gb memory and 4 processors of 2.67 GHz (Intel Core i5). We observed that using a commutative + reduced in average the number of states by a factor of 8, search time by a factor of 11, and the number of solutions by a factor of 16. The use of big-step rules, on the other hand, did not affect the number of solutions found, but reduced considerably the number of states, by a factor of 23, and search time, by a factor 40. The accumulated reduction when using both optimizations was of a factor 58 on the number of states, 118 on search time, and 16 on the number of solutions.

The Maude code for this scenario using all combinations of the two optimizations described above as well as their experimental results can be found in [22].

7 Related Work

The specification of regulations has been topic of many recent works. In [3, 4, 20], a temporal logic formalism for modeling collaborative systems is introduced. In this framework, one relates the scope of privacy to the specific roles of agents in the system. For instance, a patient's test results, which normally should not be accessible to any agent, are accessible to the agent that has the role of the patient's doctor. We believe that our system can be adapted or extended to accommodate such roles depending on the scenario considered. In particular, it also seems possible to specify in our framework the health insurance scenario discussed in [20]. De Young *et al.* describe in [8] the challenges of formally specifying the temporal properties of regulations, such as HIPAA and GLPA. They extend the temporal logic introduced in [3] with fixed point operators, which seem to be required in order to specify these regulations. A temporal logic to specify regulations, such as the FDA Code of Federal Regulations (CFR), as properties of traces abstractly representing the operations of an organization is given in [10]. Notions of permissions and obligations are introduced to deal with regulatory sentences as conditions or exceptions to others. An algorithm to check conformance of audit logs to security and privacy policies expressed in a first-order logic with restricted quantification is presented in [14]. In the case of incomplete logs a residual policy is returned.

Temporal logics are suitable for specifying the temporal properties that need to be satisfied by the traces of a system's operation. Our approach starts with an executable specification of a system using rewriting logic, combined with a mechanism to specify and check properties of executions. Specifically, critical and goal configurations defined in the equational sublogic allow us to express properties needed for generating plans for patient visits, and for monitoring clinical investigations including FDA reporting regulations. Timestamps allow us to express both temporal properties and timing constraints. Moreover, this approach allows us to use existing rewriting tools, such as Maude [6], to implement our specifications and analyses.

The Petri nets (PNs) community has investigated many related problems involving time. In particular, the coverability problem of PNs is related to our partial goal reachability problem for TLSTSes of a simple form - without branching actions, or critical states, or fresh values [16]. In [7], de Frutos Escrig *et al.* show decidability results for the coverability problem of a type of Timed PNs with discrete time. There seem to be connections between our timestamps of facts and their time (age) associated to tokens as well as connections between our time constraints and their time intervals labeling the arcs in these PNs. However, the complexity of their decision procedures is extremely high, as compared with our upper bounds. Notice that branching actions and critical states are not considered there. Despite these connections, we did not find any work that captures exactly the model presented in this paper.

Real time systems differ from our setting in that dense time domains, such as the real numbers, are required, while in our intended applications, such as clinical investigations, discrete numbers suffice. The models introduced in [1, 17, 24] deal with the specification of real time systems and also explore the complexity of some problems.

Kanovich *et al.* in [17] propose a linear logic based framework for specifying and model-checking real time systems. In particular, they demonstrate fragments of linear logic for which safety problems are PSPACE-complete. Interestingly, their examples are all balanced which is in accordance to some of our conditions. However, as discussed in [9], their model is limited since one is not allowed to specify properties which involve different timestamps. In our formalism, such properties can be specified using time constraints. In [24] conditions are identified for which the problem of checking whether a system satisfies a property, specified in linear temporal logic, is decidable. As their main application is for real time systems, they also assume dense time domains, although discrete time domains can also be accommodated. They identify non-trivial conditions on actions which allow one to abstract time and recover completeness. We are currently investigating whether a simpler definition of balanced actions and relative time constraints can provide more intuitive abstractions for systems with dense times.

Finally, there is a large body of work on Timed Automata. (See [1] for a survey.) While we extend multiset rewriting systems with time, Timed Automaton extend automaton with real-time clocks. Although timed automaton seem suitable for modeling real-time systems, such as circuits, it is not yet clear whether it is also suitable for modeling collaborative systems with explicit time. We are currently investigating connections between our formalism and Timed Automata.

8 Conclusions and Future Work

This paper introduced a model based on multiset rewriting that can be used for specifying policies and systems which mention time explicitly. We have shown that the plan compliance problem for balanced systems not containing branching actions is PSPACE-complete and the same problem for balanced systems possibly containing branching actions is EXPTIME-complete.

There are many directions which we intend to follow. In [23], we describe how an assistant can help the participants of clinical investigations to reduce mistakes and comply with policies. We are currently extending our current implementation into a small scale prototype in Maude in order to collect more feedback from the health care community. One main challenge, however, is to specify procedures in a modular fashion. One might need to specify intermediate languages that are closer to the terminology and format used in the specification of CIs, but that are still precise enough to translate them to a TLSTS. We hope that the work described in [11] may help us achieve this goal.

We would also like to extend our model to include dense times. This would allow us to specify policies for which real-times are important. For instance, [2] describes how one can reduce human errors by connecting medical devices and configuring them according to some hospital policies.

Another interesting problem to explore is checking whether a given plan, for example, a plan embedded in a protocol, complies with regulations no matter how it is executed. Such checks would help protocol design and review, and FDA audits as well as sponsors to monitor CIs and detect mistakes as early as possible.

Acknowledgments: We thank Anupam Datta, Nikhil Dinesh, Deepak Garg, Insup Lee, John Mitchell, Grigori Mints, Oleg Sokolsky, and Martin Wirsing for helpful discussions. This material is based upon work supported by the MURI program under AFOSR Grant No: FA9550-08-1-0352 and upon work supported by the MURI program under AFOSR Grant "Science of Cyber Security: Modeling, Composition, and Measurement". Additional support for Scedrov from NSF Grant CNS-0830949 and from ONR grant N00014-11-1-0555. Nigam was supported by the Alexander von Humboldt Foundation. Kanovich was partly supported by the EPSRC.

References

1 R. Alur and P. Madhusudan. Decision problems for timed automata: A survey. In *SFM*. 2004.

2 D. Arney, M. Pajic, J. M. Goldman, I. Lee, R. Mangharam, and O. Sokolsky. Toward patient safety in closed-loop medical device systems. In ICCPS '10, 2010.

3 A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *IEEE Symposium on Security and Privacy*, pages 184–198, 2006.

4 A. Barth, J. C. Mitchell, A. Datta, and S. Sundaram. Privacy and utility in business processes. In *CSF*, pages 279–294, 2007.

5 A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28:114–133, 1981.

6 M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude: A High-Performance Logical Framework*. Springer, 2007.

7 D. de Frutos Escrig, V. V. Ruiz, and O. M. Alonso. Decidability of properties of timed-arc petri nets. In *In ICATPN'00*, pages 187–206. Springer-Verlag, 2000.

8 H. DeYoung, D. Garg, L. Jia, D. K. Kaynar, and A. Datta. Experiences in the logical specification of the HIPAA and GLBA privacy laws. In *WPES*, pages 73–82, 2010.

9 H. DeYoung, D. Garg, and F. Pfenning. An authorization logic with explicit time. In *CSF*, pages 133–145, 2008.

10 N. Dinesh, A. K. Joshi, I. Lee, and O. Sokolsky. Reasoning about conditions and exceptions to laws in regulatory conformance checking. In *DEON*, pages 110–124, 2008.

11 N. Dinesh, A. K. Joshi, I. Lee, and O. Sokolsky. Permission to speak: A logic for access control and conformance. *J. Log. Algebr. Program.*, pages 50–74, 2011.

12 N. A. Durgin, P. Lincoln, J. C. Mitchell, and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

13 FDA. Code of federal regulations, Title 21, Chapter 1, Subchapter D, Part 312: Investigational new drug application. Available at `http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/ CFRSearch.cfm?CFRPart=312`.

14 D. Garg, L. Jia, and A. Datta. Policy auditing over incomplete logs: Theory, implementation and applications. In *CCS'11*, 2011.

15 M. Kanovich, T. B. Kirigin, V. Nigam, and A. Scedrov. Bounded memory Dolev-Yao adversaries in collaborative systems. In *FAST*, 2010.

16 M. Kanovich, P. Rowe, and A. Scedrov. Policy compliance in collaborative systems. In *CSF '09*.

17 M. I. Kanovich, M. Okada, and A. Scedrov. Specifying real-time finite-state systems in linear logic. *Electr. Notes Theor. Comput. Sci.*, 16(1), 1998.

18 M. I. Kanovich, P. Rowe, and A. Scedrov. Collaborative planning with confidentiality. *J. Autom. Reasoning*, 46(3-4):389–421, 2011.

19 M. I. Kanovich and J. Vauzeilles. The classical ai planning problems in the mirror of horn linear logic: semantics, expressibility, complexity. *Mathematical Structures in Computer Science*, 11(6):689–716, 2001.

20 P. E. Lam, J. C. Mitchell, and S. Sundaram. A formalization of HIPAA for a medical messaging system. In *TrustBus*, 2009.

21 J. Meseguer. Conditional Rewriting Logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

22 V. Nigam, T. B. Kirigin, A. Scedrov, C. Talcott, M. Kanovich, and R. Perovic. Timed collaborative systems. `http://www2.tcs.ifi.lmu.de/~vnigam/docs/TR-TLSTS/`, June 2011.

23 V. Nigam, T. B. Kirigin, A. Scedrov, C. Talcott, M. Kanovich, and R. Perovic. Towards an automated assistant for clinical investigations. In *International Health Informatics Symposium*, 2012.

24 P. C. Ölveczky and J. Meseguer. Abstraction and completeness for Real-Time Maude. *Electr. Notes Theor. Comput. Sci.*, 176(4):5–27, 2007.

25 S. F. Smith and C. L. Talcott. Specification diagrams for actor systems. *Higher-Order and Symbolic Computation*, 15(4):301–348, 2002.

Semantic Evaluation, Intersection Types and Complexity of Simply Typed Lambda Calculus

Kazushige Terui[1]

1 **RIMS, Kyoto University**
Kitashirakawa Oiwakecho, Sakyo-ku, Kyoto 606-8502, Japan
terui@kurims.kyoto-u.ac.jp

Abstract

Consider the following problem: given a simply typed lambda term of Boolean type and of order r, does it normalize to "true"? A related problem is: given a term M of word type and of order r together with a finite automaton D, does D accept the word represented by the normal form of M? We prove that these problems are n-EXPTIME complete for $r = 2n+2$, and n-EXPSPACE complete for $r = 2n+3$.

While the hardness part is relatively easy, the membership part is not so obvious; in particular, simply applying β reduction does not work. Some preceding works employ semantic evaluation in the category of sets and functions, but it is not efficient enough for our purpose.

We present an algorithm for the above type of problem that is a fine blend of β reduction, Krivine abstract machine and semantic evaluation in a category based on preorders and order ideals, also known as the Scott model of linear logic. The semantic evaluation can also be presented as intersection type checking.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases simply typed lambda calculus, computational complexity, denotational semantics, intersection types

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.323

Category Regular Research Paper

1 Introduction

Beta reduction vs. semantic evaluation. Let us begin with a simple puzzle. Consider the simply typed lambda calculus. For every simple type σ, let $\mathbf{N}(\sigma) := (\sigma \to \sigma) \to (\sigma \to \sigma)$. Then every Church numeral $\mathbf{n}$ has type $\mathbf{N}(\sigma)$, and the exponentiation function 2^x has type $\mathbf{N}(\sigma \to \sigma) \to \mathbf{N}(\sigma)$ for an arbitrary σ. Let $\mathbf{B} := o \to o \to o$ (where o is the base type) and $\mathbf{tt} := \lambda xy.x : \mathbf{B}$.

Now the question is as follows. Fix a type σ and a closed term M of type $\mathbf{N}(\sigma) \to \mathbf{B}$. Then what is the computational complexity of deciding whether $M\mathbf{n} =_\beta \mathbf{tt}$, when n ranges over the set $\mathbb{N}$ of natural numbers?

Since arbitrary hyperexponential functions are available in M, one might be tempted to answer that it requires hyperexponential time in n in general. Although it is true if one uses β reduction, there is actually a much more efficient algorithm, according to which it can be decided in *linear* time in n, for fixed M, with a huge coefficient depending only on σ.

Such an algorithm is most easily provided by *semantic evaluation* in the category **Set** of sets and functions. Define $[\![o]\!] = \{0,1\}$ and $[\![\sigma \to \tau]\!] = [\![\tau]\!]^{[\![\sigma]\!]}$. The denotation $[\![\mathbf{n}]\!]$ of $\mathbf{n}$ can be computed by induction on n, and each inductive step requires only constant time (which

Order	2	3	4	5	6	7
Complexity	P	PSPACE	EXPTIME	EXPSPACE	2-EXPTIME	2-EXPSPACE

Figure 1 Time-space alternation in simply typed lambda calculus.

depends on σ). We then apply the function $[\![M]\!]$ (which may be precomputed independently of n) to $[\![n]\!]$. Since the semantics is sound and $[\![\mathbf{tt}]\!] \neq [\![\mathbf{ff}]\!]$, we can check if $M\mathbf{n} =_\beta \mathbf{tt}$ by comparing $[\![M\mathbf{n}]\!]$ with $[\![\mathbf{tt}]\!]$. Clearly the total runtime is linear in n.

This demonstrates that β reduction is desperately inefficient for computing a finite-valued function in simply typed lambda calculus. It may be ascribed to the fact that β reduction is a versatile machinery independent of typing, so that it does not take advantage of the type information at all. In contrast, semantic evaluation fully exploits the type information and that is the main reason why it is so efficient for some problems.

Order and complexity. Turing to a more general situation, it has been observed by Statman [20] that the problem of deciding β-equivalence (or $\beta\eta$-equivalence) of two simply typed terms is not elementary recursive. The explosion of complexity is caused by the increase of the order of the input terms. Hence it is interesting to study the complexity of Statman's problem with restricted order. In this context, Schubert [19] has shown that the problem of deciding β-equivalence is PTIME complete for terms of order up to 2, and is PSPACE complete for terms of order up to 3, provided that one of the input terms is in normal form.

In the meantime, a curious phenomenon on the order and complexity has been observed in some extensions of simply typed lambda calculus. Goerdt and Seidl [8] have studied interpretations of higher type primitive recursive definitions (i.e. System T terms) in finite structures, and proved that terms of order 2r+2 characterize r-EXPTIME predicates, while terms of order 2r+3 characterize r-EXPSPACE predicates. Namely, time and space complexity classes *alternate* when the order increases. Similar results have been shown by Kristiansen and Voda [14, 13] for System T withour successor and by Hillebrand and Kanellakis [9] for simply typed lambda calculus with equality test. A remarkable similarity among all these works is that they employ a *hybrid* algorithm that mixes β reduction and semantic evaluation in **Set**.

About this paper. The goal of this paper is to show a similar time-space alternation result in plain simply typed lambda calculus. We consider the following decision problems (see the next section for precise definitions of order r and word type $\mathbf{W}$).

1. Bool(r): Given a closed term $M : \mathbf{B}$ of order r, does M reduce to $\mathbf{tt}$?
2. RegLang(r): Given a closed term $M : \mathbf{W}$ of order r and a nondeterministic finite automaton D, does D accept the word represented by the normal form of M? (The size of D is defined to be the number of states.)
3. Term(r): Given a closed term $M : \tau$ of order r and another term $N : \tau$ *in normal form*, do we have $M =_{\beta\eta} N$?

We prove:

▶ **Theorem 1** (Time-Space Alternation).

1. Bool$(2r+2)$ *and* RegLang$(2r+2)$ *are complete for* r-EXPTIME.
2. Bool$(2r+3)$ *and* RegLang$(2r+3)$ *are complete for* r-EXPSPACE *(see Figure 1).*

The third problem, that is essentially Schubert's problem, is subtle and will be discussed in Section 5.

To show the membership part, we basicaly follow [8, 9, 14] and employ a hybrid algorithm. However, it turns out that evaluation in the category **Set** is not efficient enough for our purpose (see Subsection 4.2). We are thus led to work in another cartesian closed category that arises by the Kleisli construction from a model of linear logic based on prime algebraic complete lattices and lub-preserving maps [10], or equivalently based on preorders and order ideals [23, 24]. It is simply called the *Scott model of linear logic* in [6]. Evaluation in this category can be expressed as type checking in an intersection type system that is essentially due to [18], and is similar to the essential type assignment system [21, 22]. Our algorithm also employs the mechanism of Krivine's abstract machine (KAM, [15]).

The rest of this paper is organized as follows. Section 2 is a preliminary, Section 3 introduces the semantics and the associated intersection type system, and Section 4 discusses some optimization techniques. Sections 5 proves the membership part of Theorem 1, while Section 6 proves the hardness part. Finally, Section 7 concludes the paper.

2 Preliminary

Given a set Q, $\sharp Q$ denotes its cardinality. Similarly, $\sharp\Gamma$ denotes the length if Γ is a list. The hyperexponential function hyp is defined by: $hyp(0,n) := n$ and $hyp(r+1,n) := 2^{hyp(r,n)}$. We let $hyp(-1,n) := \log n$ for convenience.

We write $poly(x)$ to denote a function bounded by some polynomial in x that depends on the context. The hyperexponential complexity classes are defined by r-EXPTIME := $\bigcup_{c>0}$ DTIME$[hyp(r,n^c)]$ and r-EXPSPACE := $\bigcup_{c>0}$ DSPACE$[hyp(r,n^c)]$. In particular, we have 0-EXPTIME = P and 0-EXPSPACE = PSPACE. The classes r-AEXPTIME and r-AEXPSPACE are similarly defined by means of alternating Turing machines. The following relations are fundamental:

$$r\text{-AEXPTIME} = r\text{-EXPSPACE}, \qquad r\text{-AEXPSPACE} = r+1\text{-EXPTIME}.$$

We work on the simply typed lambda calculus with a single base type o and without any term constants; the restriction to the single base type is inessential. So types are of the form o or $\sigma \to \tau$. Provided that countably many variables $x^\sigma, y^\sigma, z^\sigma, \dots$ are given for each type σ, terms are generated by the following rules, where $\overline{x} \rhd M : \tau$ indicates that M is a term of type τ whose free variables are among the list $\overline{x}$:

$$\frac{\overline{x} = x_1^{\sigma_1}, \dots, x_n^{\sigma_n}}{\overline{x} \rhd x_i : \sigma_i} \qquad \frac{\overline{x}, y^\sigma \rhd M : \tau}{\overline{x} \rhd \lambda y^\sigma.M : \sigma \to \tau} \qquad \frac{\overline{x} \rhd M : \sigma \to \tau \quad \overline{x} \rhd N : \sigma}{\overline{x} \rhd MN : \tau}$$

We often omit type superscripts and adopt the variable convention. We write $M : \sigma$ or M^σ to indicate that M is of type σ. By a *subtype* of σ, we simply mean a type occurring in σ as a subexpression (which has nothing to do with subtyping disciplines). We omit parentheses in the standard way, and write $\sigma^n \to \tau$ for $\sigma \to \cdots \sigma \to \tau$ (n times).

Data types for Boolean values, natural numbers and binary words are given by:

$$\begin{array}{lllllll}
\mathbf{B} & := & o \to o \to o & \mathbf{tt} & := & \lambda xy.x & : \mathbf{B} \\
 & & & \mathbf{ff} & := & \lambda xy.y & : \mathbf{B} \\
\mathbf{N} & := & (o \to o) \to (o \to o) & \mathbf{n} & := & \lambda fx.f^n x & : \mathbf{N} \\
\mathbf{W} & := & (o \to o)^2 \to (o \to o) & \mathbf{w} & := & \lambda f_0 f_1 x.f_{i_n}(\cdots f_{i_2}(f_{i_1}x)\cdots) & : \mathbf{W}.
\end{array}$$

where $n \in \mathbb{N}$ and $w = i_1 \cdots i_n \in \{0,1\}^*$.

Suppose that a term M has a subterm of type σ, and τ is a subtype of σ. The set of all such subtypes τ is denoted by $\mathit{Type}(M)$.

The *order* and *arity* of a type σ are inductively defined by:

$$\begin{array}{rclrcl} \mathit{Order}(o) & := & 0 & \mathit{Order}(\sigma \to \tau) & := & \max(\mathit{Order}(\sigma)+1, \tau) \\ \mathit{Arity}(o) & := & 1 & \mathit{Arity}(\sigma \to \tau) & := & \mathit{Arity}(\tau)+1. \end{array}$$

(We define $\mathit{Arity}(o) = 1$ rather than 0 just to make calculation easier.)

The *order* (resp. *arity*) of a term M is the maximal order (resp. arity) of any type in $\mathit{Type}(M)$. The *size* $|M|$ of M is the number of nodes in the term formation tree for M. $|\sigma|$ is similarly defined. We denote by $\Lambda(s, a, r)$ the set of terms of size $\leq s$, arity $\leq a$ and order $\leq r$.

Let $M : \sigma$. Our definition of $|M|$ does not take into account the size of a type occurring in it. However, this does not cause any practical problem, since the size of such a type can be bounded, up to some optimization, in terms of $|M|$, $|\sigma|$ and the order of M. In more detail, let $\tau_1 \to \tau_2 \in \mathit{Type}(M)$. Suppose that $\tau_1 \to \tau_2$ is not a subtype of σ, and that M does not contain any subterm of the form $(\lambda x.N)^{\tau_1 \to \tau_2}$ nor $N^{\tau_1 \to \tau_2} K^{\tau_1}$. In that case, we may safely replace $\tau_1 \to \tau_2$ occurring in M by o preserving the type of M. Repeat this until such a type $\tau_1 \to \tau_2$ does not exist in $\mathit{Type}(M)$. The resulting term is said to be *optimally typed* (it has to do with *principal typing*).

▶ **Lemma 2.** *Let $M : \sigma$ be an optimally typed term of arity a and of order r. Then we have $a \leq |M| + 3|\sigma|$. As a consequence, for any $\tau \in \mathit{Type}(M)$ we have $|\tau| \leq a^r \leq (|M| + 3|\sigma|)^r$.*

Hereafter we assume that terms are optimally typed. In practice, the result type σ and the order r will be fixed. Hence the actual size of M taking types into account is polynomial in $|M|$. Since we are only interested in complexity classes above P, we may safely ignore the size of a type occurring in a term.

Given a term M, we denote by $\langle\!\langle M \rangle\!\rangle_r$ the term of order r obtained by reducing all the redexes of order $> r$ in M. The following result is standard.

▶ **Lemma 3.** *Suppose that $M : \sigma$ and $\mathit{Order}(\sigma) \leq r$. If $M \in \Lambda(s, a, r+1)$, then $\langle\!\langle M \rangle\!\rangle_r \in \Lambda(2^s, a, r)$. $\langle\!\langle M \rangle\!\rangle_r$ can be computed in time $2^{O(s)}$.*

See [2] for a precise analysis of β reduction length in simply typed lambda calculus.

3 Semantic evaluation and intersection types

3.1 Category of preorders and order ideals

We now introduce a cartesian closed category $\mathbf{ScottL}_!$ (the notation is taken from [6]). While it can be directly presented as the category of prime algebraic complete lattices and Scott continuous functions, we prefer a more elementary description due to Winskel [23] (see also [24, 6]), which immediately suggests a correspondence with intersection types. We first explain, assuming some acquaintance with models of linear logic (see, e.g., [3, 17]), how it naturally arises from simple building blocks. But this part is not needed for the rest. We later describe an interpretation of simply typed lambda calculus in $\mathbf{ScottL}_!$ concretely, which can be understood without any prerequisite.

Let $\mathbf{ScottL}$ be the category in which each object is a preorder $A = (A, \sqsupseteq_A)$ and each morphism $R : A \nrightarrow B$ is an *order ideal*, that is a binary relation $R \subseteq A \times B$ such that $a' \sqsupseteq_A a \; R \; b \sqsupseteq_B b'$ implies $a' \; R \; b'$. Composition of relations is standard, while the identity $1_A : A \nrightarrow A$ on object A is given by $\sqsupseteq_A$. $\mathbf{ScottL}$ is a compact closed category with biproducts, where the dual A^*, the tensor product $A \otimes B$, the unit I, the biproduct $A \,\&\, B$ and the zero object $\top$ for $A = (A, \sqsupseteq_A)$ and $B = (B, \sqsupseteq_B)$ are given by:

$$\begin{array}{lcl lcl lcl} A^* & := & (A, \sqsubseteq_A) & A \otimes B & := & (A \times B, \sqsupseteq_A \times \sqsupseteq_B) & I & := & (\{*\}, =) \\ & & & A \,\&\, B & := & (A \uplus B, \sqsupseteq_A \uplus \sqsupseteq_B) & \top & := & (\emptyset, \emptyset), \end{array}$$

where $\uplus$ denotes the disjoint union (see, e.g., [3]).

Up to now, **ScottL** looks an obvious variant of the category **Rel** of sets and relations. A great advantage of **ScottL** is that it naturally admits a *set*-based (rather than *multiset*-based) interpretation of exponentials, in contrast to **Rel** (see [17] for discussion; see also [6] which shows that **ScottL** is the *extensional collapse* of **Rel**). This allows us to evaluate terms in finite structures.

Specifically, given a set X and a binary relation $R \subseteq A \times B$, $\mathcal{P}_f X$ denotes the set of finite subsets of X and $\mathcal{P}_f R \subseteq \mathcal{P}_f A \times \mathcal{P}_f B$ is defined by

$$Y \; \mathcal{P}_f R \; X \iff \forall \alpha \in X \; \exists \beta \in Y. \; \beta \; R \; \alpha.$$

A comonad $(!, \epsilon, \delta)$ in **ScottL** as well as Seely isomorphisms $m^2_{A,B}$, m^0 (i.e., exponential isomorphisms) are given as follows (where $A = (A, \sqsupseteq_A)$ and $R : A \twoheadrightarrow B$):

$$\begin{array}{rcll} !A & := & (\mathcal{P}_f A, \mathcal{P}_f \sqsupseteq_A) & \\ !R & := & \mathcal{P}_f R & : !A \twoheadrightarrow !B \\ \epsilon_A & := & \{(X, \alpha) : \exists \alpha'. \; X \ni \alpha' \sqsupseteq_A \alpha\} & : !A \twoheadrightarrow A \\ \delta_A & := & \{(X, \{Y_1, \ldots, Y_n\}) : X \sqsupseteq_{!A} Y_1 \cup \cdots \cup Y_n\} & : !A \twoheadrightarrow !!A \\ m^2_{A,B} & := & \{((X, Y), Z) : X \uplus Y \sqsupseteq_{!(A\&B)} Z\} & : !A \otimes !B \twoheadrightarrow !(A \& B) \\ m^0 & := & \{(*, \emptyset)\} & : I \twoheadrightarrow !\top. \end{array}$$

It is routine to verify that these data indeed define a model of linear logic (a *Seely category* [17]). Hence by the Kleisli construction, we obtain a cartesian closed category $\mathbf{ScottL}_!$.

Rather than specifying the exact structure of $\mathbf{ScottL}_!$, we will concretely describe how to interpret lambda terms in it.

Let Q be a finite set. To each type σ, we associate a poset $[\![\sigma]\!]_Q = ([\![\sigma]\!]_Q, \sqsupseteq_\sigma)$ as follows (we omit subscript Q for readability).

$$[\![o]\!] := (Q, =), \qquad [\![\sigma \to \tau]\!] := (\mathcal{P}_f [\![\sigma]\!] \times [\![\tau]\!], (\mathcal{P}_f \sqsupseteq_\sigma)^{-1} \times \sqsupseteq_\tau).$$

Concretely, we have $(X, \alpha) \sqsupseteq_{\sigma \to \tau} (Y, \beta)$ iff $\forall \gamma \in X \; \exists \delta \in Y. \; \gamma \sqsubseteq_\sigma \delta$ and $\alpha \sqsupseteq_\tau \beta$. In particular, $X \subseteq Y$ implies $(X, \alpha) \sqsupseteq_{\sigma \to \tau} (Y, \alpha)$. We occasionally write $X \to \alpha$ to denote a pair (X, α), and $(X_1, \ldots, X_n \rhd \alpha)$ to denote an $n + 1$ tuple $(X_1, \ldots, X_n, \alpha)$.

To each $\overline{x} \rhd M : \tau$ with $\overline{x} = x_1^{\sigma_1} \ldots, x_n^{\sigma_n}$, we associate a set

$$[\![\overline{x} \rhd M]\!] \subseteq \mathcal{P}_f [\![\sigma_1]\!] \times \cdots \times \mathcal{P}_f [\![\sigma_n]\!] \times [\![\tau]\!]$$

as follows.

$$\begin{array}{rcl} [\![\overline{x} \rhd x_i]\!] & := & \{(\overline{X} \rhd \alpha) : \exists \alpha' \in [\![\sigma_i]\!]. \; X_i \ni \alpha' \sqsupseteq_{\sigma_i} \alpha\} \\ [\![\overline{x} \rhd \lambda y^\sigma.M]\!] & := & \{(\overline{X} \rhd Y \to \alpha) : (\overline{X}, Y \rhd \alpha) \in [\![\overline{x}, y^\sigma \rhd M]\!]\} \\ [\![\overline{x} \rhd M^{\sigma \to \tau} N^\sigma]\!] & := & \{(\overline{X} \rhd \alpha) : \exists Y \in \mathcal{P}_f [\![\sigma]\!]. (\overline{X} \rhd Y \to \alpha) \in [\![\overline{x} \rhd M]\!] \text{ and} \\ & & \qquad \forall \beta \in Y. (\overline{X} \rhd \beta) \in [\![\overline{x} \rhd N]\!]\} \end{array}$$

We write $[\![M]\!]$ instead of $[\![\rhd M]\!]$ when M is closed. Since $\mathbf{ScottL}_!$ is cartesian closed, we have:

▶ **Theorem 4.** *If $M^\sigma =_{\beta\eta} N^\sigma$, then $[\![\overline{x} \rhd M]\!] = [\![\overline{x} \rhd N]\!]$.*

3.2 $\mathbf{ScottL_!}$ as an intersection type system

As the notation suggests, an element $X \to \alpha \in [\![\sigma \to \tau]\!]$ can be seen as an implication. When X is a set $\{\beta_1, \dots, \beta_n\}$, it is natural to think of it as an intersection type $\beta_1 \wedge \cdots \wedge \beta_n \to \alpha$.

More precisely, we rewrite a statement

$$(X_1, \dots, X_n \rhd \alpha) \in [\![x_1^{\sigma_1}, \dots, x_n^{\sigma_n} \rhd M^\sigma]\!],$$

where $X_i \in \mathcal{P}_f[\![\sigma_i]\!]$ and $\alpha \in [\![\sigma]\!]$, as a typing judgement

$$x_1^{\sigma_1} : X_1, \dots, x_n^{\sigma_n} : X_n \vdash M : \alpha.$$

The inductive definition of $[\![\overline{x} \rhd M]\!]$ may be rewritten as the following type inference rules:

$$\frac{\exists \alpha'.\ X_i \ni \alpha' \sqsupseteq_{\sigma_i} \alpha}{x_1^{\sigma_1} : X_1, \dots, x_n^{\sigma_n} : X_n \vdash x_i^{\sigma_i} : \alpha}\ (\text{var}) \qquad \frac{\Gamma, y^\sigma : Y \vdash M : \alpha}{\Gamma \vdash \lambda y^\sigma.M : Y \to \alpha}\ (\text{abs})$$

$$\frac{\Gamma \vdash M : X \to \alpha \quad \Gamma \vdash N : \beta \text{ for every } \beta \in X}{\Gamma \vdash MN : \alpha}\ (\text{app}), \quad \text{in particular} \quad \frac{\Gamma \vdash M : \emptyset \to \alpha}{\Gamma \vdash MN : \alpha}\ (\text{app})$$

$$\frac{}{\alpha \sqsupseteq_o \alpha} \qquad \frac{\forall \alpha \in X\ \exists \beta \in Y.\ \beta \sqsupseteq_\sigma \alpha}{Y \sqsupseteq_{!\sigma} X} \qquad \frac{Y \sqsupseteq_{!\sigma} X \quad \alpha \sqsupseteq_\tau \beta}{X \to \alpha \sqsupseteq_{\sigma \to \tau} Y \to \beta}$$

Since the type system comes from a cartesian closed category, typing is preserved under $\beta\eta$ equivalence. Also, $x^\sigma : X \vdash M^\tau : \alpha$, $Y \sqsupseteq_{!\sigma} X$ and $\alpha \sqsupseteq_\tau \beta$ imply $x : Y \vdash M : \beta$, since $[\![x \rhd M]\!]$ is a morphism in $\mathbf{ScottL_!}$.

▶ Remark. The above type system is essentially the same as that of [18]. It is also very close to the essential type assignment of [21], although we only deal with *simply typed* lambda terms, and thus our intersection types always conform to the shape of simple types; for instance, we do not have a type like $p \wedge (q \to r)$. It is for this reason that our types are preserved under η expansion, which does not usually hold for intersection types. It should be noted that a reversed notation is very often used for subsumption: our $a \sqsupseteq b$ corresponds to $a \leqslant b$ of [21].

It has been observed by Carvalho [4] that **Rel** leads to a (useful) intersection type system. More recently, $\mathbf{ScottL_!}$ is presented as an intersection type system by Ehrhard, that is similar to ours but for a different purpose. It could be interesting to explore this connection between relation-based semantics and intersection type systems from a more general perspective. Notice that this connection is entirely different from that between intersection types and filter models (see, e.g., [22]).

3.3 Applications

Let us illustrate the use of $\mathbf{ScottL_!}$ and the companion intersection type system by three examples.

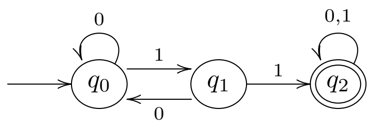
1. Suppose that $[\![o]\!] = Q = \{*\}$. Then both $\mathsf{True} := \{*\} \to \emptyset \to *$ and $\mathsf{False} := \emptyset \to \{*\} \to *$ belong to $[\![\mathbf{B}]\!]$. We have $\mathsf{True} \in [\![\mathbf{tt}]\!]$ but not $\mathsf{True} \in [\![\mathbf{ff}]\!]$. As a consequence:

▶ **Theorem 5.** *Let M be a closed term of type $\mathbf{B}$. Then $M =_{\beta\eta} \mathbf{tt}$ iff $\mathsf{True} \in [\![M]\!]$.*

In this way checking $\beta\eta$ equivalence boils down to checking membership in $\mathbf{ScottL_!}$, or equivalently to type checking in the intersection type system.

2. Let $D = (Q, \{0,1\}, \delta, q_0, q_f)$ be a nondeterministic finite automaton (NFA) where Q is the set of states, $\delta \subseteq \{0,1\} \times Q \times Q$ is the transition relation, $q_0, q_f \in Q$ are respectively

(a)



(b)

$$\dfrac{\dfrac{\dfrac{F_1 \ni \{q_1\} \to q_2}{\Gamma \vdash f_1 : \{q_1\} \to q_2} \qquad \dfrac{\dfrac{F_1 \ni \{q_0\} \to q_1}{\Gamma \vdash f_1 : \{q_0\} \to q_1} \quad \dfrac{\{q_0\} \ni q_0}{\Gamma \vdash x : q_0}}{\Gamma \vdash f_1 x : q_1}}{\Gamma \vdash f_1(f_1 x) : q_2}}{\vdash \mathbf{11} : \mathsf{D}_0}$$

Figure 2 (a) Finite automaton D_0. (b) A derivation (where $\Gamma = f_0 : F_0, f_1 : F_1, x : \{q_0\}$).

the initial and final states (we may assume w.l.o.g. that an NFA has exactly one final state). This can be expressed by an element

$$\mathsf{D} := X_0 \to X_1 \to q_0 \to q_f \quad \in [\![\mathbf{W}]\!]_Q,$$

where $X_0 = \{\{q_i\} \to q_j : (0, q_i, q_j) \in \delta\}$, and $X_1 = \{\{q_i\} \to q_j : (1, q_i, q_j) \in \delta\}$.

For instance, to the automaton $D_0 = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, q_2)$ described in Figure 2 (a) corresponds the element $\mathsf{D}_0 := F_0 \to F_1 \to \{q_0\} \to q_2$, where $F_0 = \{\{q_0\} \to q_0, \{q_1\} \to q_0, \{q_2\} \to q_2\}$ and $F_1 = \{\{q_0\} \to q_1, \{q_1\} \to q_2, \{q_2\} \to q_2\}$.

Corresponding to the run accepting the word 11, we have $\mathsf{D}_0 \in [\![\mathbf{11}]\!]$, i.e., $\vdash \mathbf{11} : \mathsf{D}_0$, as demonstrated by the derivation in Figure 2 (b); recall that $\mathbf{11} = \lambda f_0 f_1 x. f_1(f_1 x)$. Conversely, given a derivation for $\vdash \mathbf{w} : \mathsf{D}_0$, we can read off a successful run on w. Hence we have:

▶ **Theorem 6.** *Let D be a nondeterministic finite automaton and $w \in \{0, 1\}^*$. Then D accepts w iff $\mathsf{D} \in [\![\mathbf{w}]\!]$. Hence if M is a closed term of type $\mathbf{W}$, D accepts the word represented by the normal form of M iff $\mathsf{D} \in [\![M]\!]$.*

▶ Remark. We can conversely show that every element of $[\![\mathbf{w}]\!]$ corresponds to an NFA accepting w. So the set $\bigcup_{Q:\text{finite}} [\![\mathbf{w}]\!]_Q$ can be thought of as the set of NFA's accepting w.

The same idea is behind Kobayashi's use of intersection types for model checking higher order recursion schemes [11]. Adapted to the current setting, it amounts to the following. Let M be a closed term of infinite tree type $\mathbf{T}^\omega := (o^2 \to o)^2 \to o$, involving the fixpoint operator μ. Then M possibly describes an infinite binary tree $\langle\!\langle M \rangle\!\rangle$ labelled with $\{0, 1\}$. For every tree automaton D with a trivial acceptance condition [1], there is an element $\mathsf{D} \in [\![\mathbf{T}^\omega]\!]$ such that $\langle\!\langle M \rangle\!\rangle$ is accepted by D iff $\mathsf{D} \in [\![M]\!]$.

The intersection type system of [11] is later expanded by [12] so that the new type system covers all the properties expressible by modal μ-calculus formulas. It would be interesting to see whether the latter can also be seen as a model of lambda calculus like $\mathbf{ScottL}_!$.

3. Given a closed term N in η-long normal form, let $Q = \{\mathsf{L}_1, \ldots, \mathsf{L}_m\}$ be the set of symbols, each L_i corresponding to a subterm occurrence of base type o in N. We can then associate an element $\mathsf{M} \in [\![\sigma]\!]_Q$ to each subterm occurrence M^σ in N by induction on the structure of M. If $M = xK_1 \cdots K_j$ that is a subterm of $L_i^o = xK_1 \cdots K_n$ $(0 \leq j < n)$, we define $\mathsf{M} := \{\mathsf{K}_{j+1}\} \to \cdots \{\mathsf{K}_n\} \to \mathsf{L}_i$. In particular, $\mathsf{x} := \{\mathsf{K}_1\} \to \cdots \{\mathsf{K}_n\} \to \mathsf{L}_i$. If $M = \lambda x.K$ and $x^{(1)}, \cdots, x^{(n)}$ are the occurrences of variable x in K, we define $\mathsf{M} := \{\mathsf{x}^{(1)}, \cdots, \mathsf{x}^{(n)}\} \to \mathsf{K}$.

For instance, consider the Kirstead terms:

$$K_1 := \lambda f. f(\lambda y. f(\lambda z. y)), \qquad K_2 := \lambda f. f(\lambda y. f(\lambda z. z)) \qquad : \ ((o \to o) \to o) \to o.$$

Let $F_1 := f(\lambda z.y), G_1 := f(\lambda y.f(\lambda z.y)), F_2 := f(\lambda z.z)$ and $G_2 := f(\lambda y.f(\lambda z.z))$. We have

$$\begin{array}{lcl} \mathsf{K}_1 & := & \{\{\emptyset \to \mathsf{y}\} \to \mathsf{F}_1, \{\{\mathsf{y}\} \to \mathsf{F}_1\} \to \mathsf{G}_1\} \to \mathsf{G}_1, \\ \mathsf{K}_2 & := & \{\{\{z\} \to \mathsf{z}\} \to \mathsf{F}_2, \{\emptyset \to \mathsf{F}_2\} \to \mathsf{G}_2\} \to \mathsf{G}_2. \end{array}$$

We can verify that these elements distinguish K_1 from K_2, as $\vdash K_i : \mathsf{K}_j$ holds iff $i = j$. More generally, we have the following theorem, which is due to Salvati [18].

▶ **Theorem 7.** *Let N be a closed term of type σ in η-long normal form. For every closed term K of type σ, $K =_{\beta\eta} N$ iff $\mathsf{N} \in [\![K]\!]$.*

As a consequence, interpretation of lambda terms in $\mathbf{ScottL_!}$ is *injective*:

▶ **Corollary 8.** *Let M and N be closed terms of the same type. Then $M =_{\beta\eta} N$ iff $[\![M]\!]_Q = [\![N]\!]_Q$ holds for all finite sets Q iff $[\![M]\!]_Q = [\![N]\!]_Q$ holds for an infinite set Q.*

Notice that injectivity also holds for **Set** [7], **Rel** and **Coh** (the category of coherent spaces and stable maps) [5].

4 Optimizations

4.1 Krivine type system

The type inference rules in the previous subsection suggest a natural alternating algorithm for type checking. For instance, the rule (app) says that $\Gamma \vdash MN : \alpha$ holds iff there *exists* a set X such that $\Gamma \vdash M : X \to \alpha$ and for *every* $\beta \in X$, $\Gamma \vdash N : \beta$. Unfortunately, the algorithm is not efficient enough. The reason is that, when applying the rule (app) bottom-up, we have to guess a set X, which is sometimes too big. To avoid a too big guess, we incorporate the mechanism of Krivine's abstract machine [15] into the type system.

Throughout this subsection, we fix a set Q, $r \in \mathbb{N}$, and only consider types and terms of order $\leq r+1$. First, for each simple type σ, let $CL(\sigma)$ be the set $\{(n, N) : n \in \mathbb{N},\ N : \sigma\}$ of *closures*. Next, notice that each simple type σ of order $\leq r+1$ can be uniquely written as $\sigma = \sigma_1 \to \cdots \sigma_k \to \tau$ with $Order(\sigma_k \to \tau) = r+1$ and $Order(\tau) \leq r$ ($k = 0$ if $Order(\sigma) \leq r$). We define the set $ST(\sigma)$ of *stacks* by:

$$ST(\sigma) := \{(n_1, N_1) \to \cdots (n_k, N_k) \to \alpha : (n_i, N_i) \in CL(\sigma_i) \text{ for } 1 \leq i \leq k \text{ and } \alpha \in [\![\tau]\!]\}.$$

An *environment* Γ is a list of the form $x_1^{\sigma_1} : C_1, \ldots, x_n^{\sigma_n} : C_n$ with $Order(\sigma_i) \leq r$ such that $C_i \in CL(\sigma_i)$ if $Order(\sigma_i) = r$ and $C_i \in CL(\sigma_i) \cup \mathcal{P}_f[\![\sigma_i]\!]$ if $Order(\sigma_i) < r$ for every $1 \leq i \leq n$. A *judgement* is of the form $\Gamma \vdash_r M : S$ with $M : \sigma$, $Order(\sigma) \leq r+1$ and $S \in ST(\sigma)$. Here we put subscript r to emphasize that the definition depends on the order r.

The following rules define the *Krivine type system*:

$$\frac{\exists \alpha'.\ X \ni \alpha' \sqsupseteq_\sigma \alpha}{\Delta, x^\sigma : X, \Sigma \vdash_r x^\sigma : \alpha}\ (\text{var } 1) \qquad \frac{\Gamma \vdash_r N : \alpha}{\Gamma, \Delta, x^\sigma : (\sharp\Gamma, N), \Sigma \vdash_r x^\sigma : \alpha}\ (\text{var } 2)$$

$$\frac{\Gamma, x^\sigma : X \vdash_r M : \alpha}{\Gamma \vdash_r \lambda x^\sigma.M : X \to \alpha}\ (\text{abs } 1) \qquad \frac{\Gamma, x^\sigma : (n, N) \vdash_r M : S}{\Gamma \vdash_r \lambda x^\sigma.M : (n, N) \to S}\ (\text{abs } 2)$$

$$\frac{\Gamma \vdash_r M : X \to \alpha \quad \Gamma \vdash_r N : \beta\ (\forall \beta \in X)}{\Gamma \vdash_r MN : \alpha}\ (\text{app } 1) \qquad \frac{\Gamma \vdash_r M : (\sharp\Gamma, N) \to S}{\Gamma \vdash_r MN : S}\ (\text{app } 2)$$

Note that the system is syntax-directed, in the sense that at most one rule can be applied bottom-up to each judgement.

The main purpose of the above system is to avoid guessing a set $X \in \mathcal{P}_f[\![\sigma]\!]$ when $Order(\sigma) = r$ in the case of application. We instead invoke a KAM computation. Judgements correspond to KAM states, and rules (var 2), (abs 2) and (app 2) correspond to KAM transition rules. Indeed, when read bottom-up, (app 2) forms a closure (Γ, N) and pushes it to the stack, and (abs 2) pops a closure (Γ, N) from the stack and updates the environment with $x : (\Gamma, N)$. Finally, (var 2) looks up the value of x in the current environment.

The only difference is that our closure does not record an environment Γ itself, but only its length $\sharp\Gamma$. Due to our restricted use of KAM, the former can be recovered from the latter. Formally, let us call a judgement $\Gamma \vdash_r M : S$ *well-formed* when the following holds:
1. If $\Gamma = \Delta, x : (n, N), \Sigma$ then $n \leq \sharp\Delta$.
2. If (n, N) occurrs in S, then $n \leq \sharp\Gamma$.
One can immediately see that if a well-formed judgement is derivable, then it always has a derivation in which all judgements are well-formed.

The two derivations below, the left one in the previous system and the right one in the new system, illustrate how the KAM machinery works effectively (recall that $\Gamma \vdash N : \beta$ and $\beta \sqsupseteq \beta'$ imply $\Gamma \vdash N : \beta'$):

$$\dfrac{\dfrac{\begin{array}{c}\dfrac{X \ni \beta \sqsupseteq \beta'}{\Gamma, x : X, \Delta \vdash x : \beta'} \\ \vdots \\ \Gamma, x : X \vdash M : \alpha\end{array}}{\Gamma \vdash \lambda x.M : X \to \alpha} \quad \begin{array}{c}\vdots \\ \Gamma \vdash N : \beta \ (\forall \beta \in X)\end{array}}{\Gamma \vdash (\lambda x.M)N : \alpha} \quad \Longrightarrow \quad \dfrac{\dfrac{\begin{array}{c}\dfrac{\begin{array}{c}\vdots \\ \Gamma \vdash_r N : \beta'\end{array}}{\Gamma, x : (\sharp\Gamma, N), \Delta \vdash_r x : \beta'} \\ \vdots \\ \Gamma, x : (\sharp\Gamma, N) \vdash_r M : \alpha\end{array}}{\Gamma \vdash_r \lambda x.M : (\sharp\Gamma, N) \to \alpha}}{\Gamma \vdash_r (\lambda x.M)N : \alpha}$$

▶ **Theorem 9.** *Let $M : \sigma$ be a closed term of order $r + 1$ and $\alpha \in [\![\sigma]\!]$. Then $\alpha \in [\![M]\!]$ iff $\vdash_r M : \alpha$ is derivable in the Krivine type system.*

Proof. Recall that $\alpha \in [\![M]\!]$ iff $\vdash M : \alpha$ holds in the intersection type system in Subsection 3.2. For the forward direction, we tentatively work in an intermediate system in which the definition of $ST(\sigma)$ is relaxed to $ST(\sigma) := \{C_1 \to \cdots C_n \to \alpha : C_i \in CL(\sigma_i) \cup \mathcal{P}_f[\![\sigma_i]\!]$ for $1 \leq i \leq n$ and $\alpha \in [\![o]\!]\}$ for $\sigma = \sigma_1 \to \cdots \sigma_n \to o$. Suppose that $\Gamma \vdash N : \beta$ holds for every $\beta \in X$. We then have:
1. If $\Gamma, \Delta, x : X, \Sigma \vdash M : S$ is derivable in the intermediate system, then so is $\Gamma, \Delta, x : (\sharp\Gamma, N), \Sigma \vdash M : S$.
2. If $\Gamma, \Delta \vdash M : \overline{C} \to X \to S$ is derivable in the intermediate system (where $\overline{C}$ is a list of closures/intersection types), then so is $\Gamma, \Delta \vdash M : \overline{C} \to (\sharp\Gamma, N) \to S$.
We can now replace each instance of (app 1) in a derivation with that of (app 2). Indeed, if we have $\Gamma \vdash M : X \to S$ and $\Gamma \vdash N : \beta$ for every $\beta \in X$, then $\Gamma \vdash M : (\sharp\Gamma, N) \to S$ holds by 2 above, so $\Gamma \vdash MN : S$ holds by (app 2).

For the backward direction, we evaluate terms M in environments Γ. Given an environment Γ and $k \leq \sharp\Gamma$, we write Γ_k to denote its initial segment of length k, and $\hat{\Gamma}$ to denote Γ with all members of the form $x : (n, N)$ removed. We define evaluation $M[\Gamma]$ by:

$$M[\emptyset] := M, \qquad M[\Gamma, x : X] := M[\Gamma], \qquad M[\Gamma, x : (k, N)] := (M[\Gamma])[\,N[\Gamma_k]\,/x],$$

where $[N/x]$ denotes the standard substitution. We then have the following property.

- If a well-formed judgement $\Gamma \vdash_r M : (n_1, N_1) \to \cdots M : (n_k, N_k) \to \alpha$ is derivable in the Krivine type system, then $\hat{\Gamma} \vdash M[\Gamma]N_1[\Gamma_{n_1}] \cdots N_k[\Gamma_{n_k}] : \alpha$ is derivable in the previous type system.

The proof employs the fact that typing is preserved by β equivalence. ◀

4.2 Thin types

To give an efficient algorithm for problems $\textsc{Bool}(r)$ and $\textsc{RegLang}(r)$, it is useful to restrict the set of elements in $[\![\sigma]\!]$ (= intersection types) slightly. Let $\sigma = \sigma_1 \to \cdots \sigma_m \to o^k \to o$

with $\sigma_m \neq o$ (or $\sigma = o^k \to o$ and $m = 0$). An intersection type $\beta = X_1 \to \cdots X_{m+k} \to \alpha$ in $[\![\sigma]\!]$ is said to be *thin* if $\biguplus_{1\leq i\leq k} X_{m+i}$ is either empty or a singleton, and $X_1, \dots, X_m$ consist of thin types (whenever $m > 0$). The set of all thin elements in $[\![\sigma]\!]$ is denoted by $[\![\sigma]\!]^*$.

For instance, both $\mathsf{True} = \{*\} \to \emptyset \to *$ and $\mathsf{False} = \emptyset \to \{*\} \to *$ are thin, while $\{*\} \to \{*\} \to *$ is not. The type $\mathsf{D} \in [\![\mathbf{W}]\!]$ for a finite automaton D is also thin.

▶ **Lemma 10.** *Let σ be either $\mathbf{B}$ or $\mathbf{W}$. Then for every closed term $M : \sigma$ and $\alpha \in [\![\sigma]\!]^*$, $\vdash_r M : \alpha$ holds if and only if it has a thin derivation (i.e., a derivation which involves only thin types).*

Proof. We prove the lemma for the intersection type system in Subsection 3.2. We can then transform the derivation into that of the Krivine type system without introducing a non-thin type (see the proof of Theorem 9). For simplicity, we only consider the case $\sigma = \mathbf{B}$.

Suppose that $\vdash M : \mathbf{B}$ holds but it does not admit a thin derivation. By strong nomalization, we may assume that M has exactly one redex $(\lambda x_1.K)N_1$, and its reduct admits a thin derivation. Observe that $\lambda x_1.K$, which is responsible for non-thinness, is of type $o^k \to o$, and that substituting N_1 for the variable x_1 of type o does not create a new redex in K. As a consequence, M actually contains a generalized redex $(\lambda x_1^o \cdots x_k^o.K_0)N_1 \cdots N_l$ $(1 \leq l \leq k)$, and it reduces to a subterm of $\mathbf{tt}$ or $\mathbf{ff}$ after l steps of β reduction. Hence $K_0, N_1, \dots, N_l$ are variables of type o. But then we can directly verify that $(\lambda x_1^o \cdots x_k^o.K_0)N_1 \cdots N_l$ admits a thin derivation, contradicting the assumption. ◀

▶ **Lemma 11.** *Let Q be a finite set such that $\sharp Q = q$, σ be a simple type of order $r \geq 1$ and of arity a. Then $\sharp[\![\sigma]\!]^*_Q \leq hyp(r-1, O(a^2q^2))$. Hence each element in $[\![\sigma]\!]^*_Q$ can be effectively coded by a string of length $hyp(r-2, O(a^2q^2))$.*

Proof. If $r = 1$, then σ is of the form $o^k \to o$. Since we only count thin types, $\sharp[\![\sigma]\!]^* = kq^2 + q \leq aq^2$. If $r = 2$ and $\sigma = \sigma_1 \to \cdots \sigma_k \to o$, then

$$\sharp[\![\sigma]\!]^* \;\leq\; 2^{\sharp[\![\sigma_1]\!]^* + \cdots + \sharp[\![\sigma_k]\!]^*} \cdot q \;\leq\; 2^{(a-1)(aq^2)+\log q} \;\leq\; 2^{a^2q^2}.$$

The rest can be calculated by induction. ◀

▶ Remark. If we naively interpret types in **Set**, the upper bound on the number of elements in $[\![\sigma]\!]$ would be something like: q^{q^a} for $Order(\sigma) = 1$, and $q^{q^{q^{a \log a}}}$ for $Order(\sigma) = 2$, that is too much sensitive to the arity a. This is the main reason why we work with $\mathbf{ScottL}_!$ rather than **Set**.

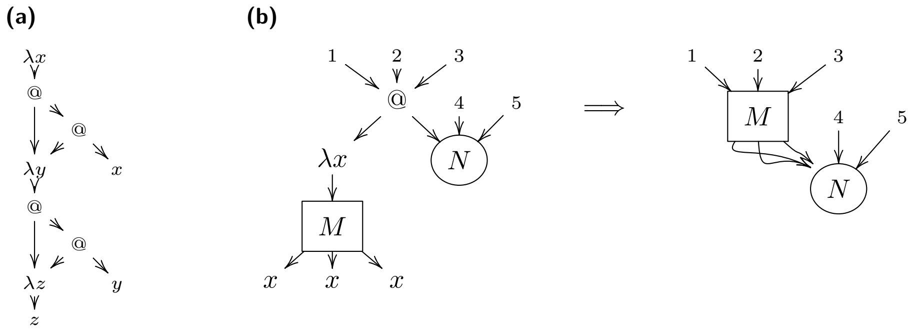
Restriction to thin elements only works for "unary" data types such as $\mathbf{B}$, $\mathbf{N}$ and $\mathbf{W}$; typically, it does not work for tree type $\mathbf{T} := (o^2 \to o)^2 \to o^2 \to o$, since tree automata cannot be expressed by thin types.

5 Complexity of evaluation

5.1 Order $2r + 3$

We are now ready to estimate the complexity of semantic evaluation in $\mathbf{ScottL}_!$.

▶ **Theorem 12.** *Let $M : \sigma$ be a closed term in $\Lambda(s, a, r + 3)$, where σ is either $\mathbf{B}$ or $\mathbf{W}$ and $r \geq 0$. Let $\alpha \in [\![\sigma]\!]^*_Q$ and $q = \sharp Q$. Then it can be decided by an alternating algorithm in time $s^3 \cdot hyp(r, O(a^2q^2))$ whether $\vdash_{r+2} M : \alpha$ holds or not.*



■ **Figure 3** (a) A dag-representation of $\lambda x.(\lambda y.(\lambda z.z)((\lambda z.z)y))((\lambda y.(\lambda z.z)((\lambda z.z)y))x)$.
(b) A graph rewriting rule corresponding to $(\lambda x.M)N \longrightarrow M[N/x]$.

Proof. We apply the natural alternating algorithm inherent to the definition of the Krivine type system, where stacks are built upon thin types.

It is clear that the height of any derivation is at most $s + ar$, where ar takes into account the height of subderivations for subsumption. Each step of applying a rule (bottom up) costs linear time in the size of a judgement. Hence it suffices to give an upper bound to the size of any judgement occurring in an (attempted) derivation of $\vdash_{r+2} M : \alpha$. Let

$$x_1^{\sigma_1} : C_1, \cdots, x_m^{\sigma_m} : C_m \vdash_{r+2} N^{\sigma} : C_{m+1} \to \cdots C_n \to \alpha$$

be such a judgement, where $\sigma = \sigma_{m+1} \to \cdots \sigma_n \to o$ and $Order(\sigma) \leq r+3$. We have either $C_i \in \mathcal{P}_f[\![\sigma_i]\!]^*$ with $Order(\sigma_i) \leq r+1$, or $C_i = (k, K) \in CL(\sigma_i)$. In the former case, C_i can be coded by a string of length $hyp(r, O(a^2q^2))$, while in the latter case the size is at most $2s$. Since n is bounded by $s + a$, the total size of the judgement is $(s+a) \cdot \max(hyp(r, O(a^2q^2)), 2s) + s = s^2 \cdot hyp(r, O(a^2q^2))$ (note that $a = O(s)$ by Lemma 2).

Therefore, the algorithm runs in time $s^3 \cdot hyp(r, O(a^2q^2))$. ◀

▶ **Corollary 13.** *Both* Bool$(2r+3)$ *and* RegLang$(2r+3)$ *belong to* r-EXPSPACE.

Proof. Let us consider Bool$(2r+3)$. Given an (optimally typed) closed term $M : \mathbf{B}$ of size s and of order $2r+3$, we compute $M' = \langle\!\langle M \rangle\!\rangle_{r+3}$, and check if $\vdash_{r+2} M' : \mathsf{True}$ with $Q = \{*\}$. The algorithm is correct since we have $M =_{\beta\eta} \mathbf{tt}$ iff $M' =_{\beta\eta} \mathbf{tt}$ iff $\mathsf{True} \in [\![M']\!]$ iff $\vdash_{r+2} M' : \mathsf{True}$ by Theorems 5 and 9.

The cost of computing M' is $hyp(r, O(s))$ and $M' \in \Lambda(hyp(r,s), a, r+3)$ by Lemma 3. Type checking of $\vdash_{r+2} M' : \mathsf{True}$ costs alternating time $hyp(r,s)^3 \cdot hyp(r, O(a^2)) = hyp(r, poly(s))$ by Theorem 12. Hence the whole algorithm works in alternating time $hyp(r, poly(s))$, so in deterministic space $hyp(r, poly(s))$.

The problem RegLang$(2r+3)$ can be decided similarly; this time the polynomial also depends on q, the number of states of the input automaton D (see Theorem 6). ◀

5.2 Order $2r+2$

Let us now address the problems Bool$(2r+2)$ and RegLang$(2r+2)$. Here a more delicate space management is required. In particular, type checking has to be done with dag-representations of terms (see Figure 3(a)).

A term M is of *dag-size* s, if M can be represented as a directed acyclic graph which has s vertices. Let us denote by $\Lambda^{\mathsf{dag}}(s, a, r)$ the set of terms of dag-size $\leq s$, arity $\leq a$ and order $\leq r$. Lemma 3 can be refined as follows:

▶ **Lemma 14.** *Suppose that $M : \sigma$ and $Order(\sigma) \leq r$. If $M \in \Lambda(s, a, r+1)$, then $\langle\langle M \rangle\rangle_r \in \Lambda^{\mathsf{dag}}(s, a, r)$. Given an ordinary representation of M, a dag-representation of $\langle\langle M \rangle\rangle_r$ can be computed in time $O(s^2)$.*

Proof. Represent M as a tree, and successively apply the graph rewriting rule of Figure 3(b) (that corresponds to β reduction) to each "redex" of order $r+1$. The rule is sound, provided that (*) the indicated vertex λx (of order $r+1$) does not have any incoming edge other than the indicated one from @. This property (*) is indeed maintained during the whole process of reduction, since the reduction rule applied to a redex of order $r+1$ never creates a vertex of order $r+1$ with multiple incoming edges. Since the reduction strictly decreases the number of vertices, the computation terminates in s steps, so in time $O(s^2)$. ◀

Below, we suppose that the Krivine type system operates on terms represented as dags.

▶ **Theorem 15.** *Let $M : \sigma$ be a closed term in $\Lambda^{\mathsf{dag}}(s, a, r+3)$, where σ is either $\mathbf{B}$ or $\mathbf{W}$, and $r \geq 0$. Let $\alpha \in [\![\sigma]\!]$ and $q = \sharp Q$. Then it can be decided by an alternating algorithm in space $s^2 \cdot hyp(r, O(a^2q^2))$ whether $\vdash_{r+2} M : \alpha$ holds or not.*

Proof. It suffices to show that each judgement $\Gamma \vdash N : S$ occurring in the derivation has size $s^2 \cdot hyp(r, O(a^2q^2))$. The only delicate point is to show that the length of Γ is bounded by s, the *dag-size* of M. To see this, notice that we can identify a subterm of M with a vertex in the dag-representation of M. Then a thread of a derivation in the type system of Subsection 3.2 corresponds to a path in the dag. Now the length of an environment increases only when a vertex with label λx is visited, but the same vertex cannot be visited twice due to acyclicity. Hence it is bounded by s. The derivation can be then translated into one in the Krivine type system keeping the length bound (see the proof of Theorem 9). ◀

▶ **Corollary 16.** *Both* Bool$(2r+2)$ *and* RegLang$(2r+2)$ *belong to* r-EXPTIME.

Proof. Let us consider Bool$(2r+2)$. Since the case $r = 0$ can be easily decided by simple graph rewriting [19], we assume $r = r' + 1$, so that $2r + 2 = 2r' + 4$.

Given a closed term $M : \mathbf{B}$ of size s and of order $2r'+4$, we first compute $M' := \langle\langle M \rangle\rangle_{r'+4}$, then a dag representation of $M'' := \langle\langle M' \rangle\rangle_{r'+3}$, and check if $\vdash_{r'+2} M'' : \mathsf{True}$ with $Q = \{*\}$. The cost of computing M' is as before, while M'' can be computed in space $hyp(r', O(s^2))$, and $M'' \in \Lambda^{\mathsf{dag}}(hyp(r', s), a, r'+3)$ by Lemma 14. Type checking of $\vdash_{r'+2} M'' : \mathsf{True}$ costs alternating space $hyp(r', s)^2 \cdot hyp(r', O(a^2)) = hyp(r', poly(s))$ by Theorem 15. Hence the whole algorithm works in alternating space $hyp(r', poly(s))$, so in deterministic time $hyp(r, poly(s))$. ◀

▶ Remark. Theorem 15 nearly (though not exactly) conforms to the puzzle at the beginning of the introduction. The trick there is indeed that the arity is constant for all $M\mathbf{n}$ when M is fixed, so that the number $hyp(r, O(a^2q^2))$ in Theorem 15 becomes a constant. Hence the computation can be done in alternating space $O(s^2)$, that is in deterministic time $2^{O(s)}$.

Finally, let us discuss the problem Term(r). Since we cannot assume that types are thin in general, the upper bounds on the number and size of intersection types get worse than estimated by Lemma 11. Namely, if σ is of order $r \geq 1$ and arity a, then $\sharp[\![\sigma]\!] \leq hyp(r, O(aq))$, and the size of each $\alpha \in [\![\sigma]\!]$ is $hyp(r-1, O(aq))$.

In addition, the Krivine type system $\vdash_r$ does not work when the order of the type of input terms M, N is greater than $r+1$. Hence natural complexity bounds are as follows.

▶ **Theorem 17.** *Under the assumption that the type of input terms M, N is of order $\leq r+1$:*

1. Term$(2r+1)$ *belongs to* r-EXPTIME.

2. $\textsc{Term}(2r+2)$ *belongs to* r-EXPSPACE.

It can be proved by employing Theorem 7; just notice that q is bounded by the size of the η-long normal form of N.

▶ Remark. Schubert [19] proved that $\textsc{Term}(2)$ is complete for P and $\textsc{Term}(3)$ is complete for PSPACE, which are sharper than our results. We leave it open whether we may extend the sharper bounds of [19] to higher order.

6 Hardness

6.1 Encoding of Turing machines

Let us outline a proof of the hardness part of Theorem 1. For orders 2 and 3, it can be easily proved by encoding Boolean circuits into order 2 terms, and quantified Boolean circuits into order 3 terms, respectively.

$$\begin{array}{lcll} \neg & := & \lambda bxy.byx & : \mathbf{B} \to \mathbf{B} \\ \wedge & := & \lambda b_1 b_2 xy.b_1(b_2xy)y & : \mathbf{B}^2 \to \mathbf{B} \\ \forall & := & \lambda F.F\mathbf{tt} \wedge F\mathbf{ff} & : (\mathbf{B} \to \mathbf{B}) \to \mathbf{B} \end{array}$$

Beyond order 3, we encode Turing machines with bounded time and space. In contrast to [8, 9, 14], which represent each TM by a single program, we just need to encode a TM by a *family* of lambda terms, one for each input length. A similar encoding was given by [16], but ours is more efficient with respect to the order (though less uniform). Encoding is even easier if we have product types. So we first outline an encoding with product types, and then explain how to eliminate them in the next subsection.

In the following, we write $\sigma \times \tau$ for a product type, $\langle M^\sigma, N^\tau \rangle^{\sigma\times\tau}$ for a pair, and $\pi_i(M^{\sigma_1\times\sigma_2})^{\sigma_i}$ for a projection ($i = 1, 2$). The definition of order is extended by $\mathit{Order}(\sigma \times \tau) := \max(\mathit{Order}(\sigma), \mathit{Order}(\tau))$.

Define $\mathbf{N}_0 := \mathbf{N}$ and $\mathbf{N}_{r+1} := \mathbf{N}_r \to \mathbf{N}_r$, and let $\mathbf{N}_r(\sigma)$ be $\mathbf{N}_r$ with the base type o replaced by σ. Observe that $\mathit{Order}(\mathbf{N}_r(\sigma)) = r + 2 + \mathit{Order}(\sigma)$. A natural number $k = 2^n$ can be succinctly represented by $\mathbf{n2} : \mathbf{N}_r$ with $\mathbf{n} : \mathbf{N}_{r+1}$ and $\mathbf{2} : \mathbf{N}_r$, or by $\mathbf{2} \circ \cdots \circ \mathbf{2} : \mathbf{N}_r$ (n times) with $\mathbf{2} : \mathbf{N}_r$ (where $N \circ M := \lambda z.N(Mz)$ with z fresh). Hence the number $\mathit{hyp}(r, n)$ can be represented by

$$\mathbf{hyp}(r, n) := (\underbrace{\mathbf{2} \circ \cdots \circ \mathbf{2}}_{n \textit{ times}})\ \underbrace{\mathbf{2} \cdots \mathbf{2}}_{r-1 \textit{ times}} \quad : \ \mathbf{N}_0(\sigma)$$

for every type σ, where the highest order subterms are $\mathbf{2} \circ \cdots \circ \mathbf{2}$ ($k \le n$ times) of type $\mathbf{N}_{r-1}(\sigma)$, whose order is $r + 1 + \mathit{Order}(\sigma)$.

Let $\mathbf{B_n} := o^n \to o$ and $\mathbf{i} := \lambda x_1 \cdots x_n.x_i : \mathbf{B_n}$. We have $\mathbf{B} = \mathbf{B_2}$.

For every $r, n \ge 1$, define a type $\mathbf{Addr}_{r,n}$ (for *addresses*) of order r by:

$$\mathbf{Addr}_{1,n} := \mathbf{B} \times \cdots \times \mathbf{B} \ (n \text{ times}), \qquad \mathbf{Addr}_{r+1,n} := \mathbf{Addr}_{r,n} \to \mathbf{B}.$$

Each $\mathbf{Addr}_{r,n}$ represents the set of natural numbers below $\mathit{hyp}(r, n)$ in binary. For $r = 1$, $k < 2^n$ is represented by $\mathbf{k}_{1,n} := \langle \mathbf{i_1}, \cdots, \mathbf{i_n} \rangle : \mathbf{Addr}_{1,n}$, where $i_1 \cdots i_n$ is the binary representation of k. For $r = r' + 1$, $k < \mathit{hyp}(r, n)$ is represented by the closed normal term $\mathbf{k}_{r,n} : \mathbf{Addr}_{r,n}$ such that $\mathbf{k}_{r,n}\mathbf{i}_{r',n}$ gives the ith bit of k for every $i < \mathit{hyp}(r', n)$.

▶ **Lemma 18.** *For every* $r, n \ge 1$, *there are closed terms*

$$\mathbf{suc}_{r,n},\ \mathbf{pred}_{r,n} : \mathbf{Addr}_{r,n} \to \mathbf{Addr}_{r,n}, \qquad <_{r,n} : \mathbf{Addr}_{r,n} \times \mathbf{Addr}_{r,n} \to \mathbf{B}$$

of size $O(n^2)$ *and order* $2r$, *representing successor and predecessor modulo* $\mathit{hyp}(r, n)$ *and* $<$.

Proof. It is easy for $r = 1$. For $r = r' + 1$, we first build a term $G_{f,a_0} : \mathbf{B}$ with free variables $f : \mathbf{Addr}_{r,n}$ and $a_0 : \mathbf{Addr}_{r',n}$, which computes the carry to the a_0th bit when computing $f + 1$. The idea is to iterate the step function $F_{f,a_0} : \mathbf{B} \times \mathbf{Addr}_{r',n} \to \mathbf{B} \times \mathbf{Addr}_{r',n}$ defined by $F_{f,a_0} := \lambda ca.\ \mathbf{if}\ a <_{r',n} a_0\ \mathbf{then}\ \langle c \wedge f(a), \mathbf{suc}_{r',n}(a)\rangle\ \mathbf{else}\ \langle c, a\rangle$. We thus define $G_{f,a_0} := \pi_1(\mathbf{hyp}(r', n)\ F_{f,a_0}\ \langle \mathbf{1}, \mathbf{0}_{r',n}\rangle)$, and finally $\mathbf{suc}_{r,n} := \lambda f a_0.f(a_0) \oplus G_{f,a_0} : \mathbf{Addr}_{r,n} \to \mathbf{Addr}_{r,n}$. The highest order subterm is $\mathbf{hyp}(r', n) : \mathbf{N}_0(\mathbf{Addr}_{r',n} \times \mathbf{B})$, which is of order $r' + 1 + r' \leq 2r$. $\mathbf{pred}_{r,n}$ and $<_{r,n}$ are defined similarly. ◀

▶ **Theorem 19.** *Let TM be a deterministic Turing machine with 1 tape and q states which works in space $hyp(r, n)$ and in time $hyp(r', n)$ with $1 \leq r \leq r'$. Then there exists a term $\mathbf{tm}_{r,r',n} : \mathbf{B}^n \to \mathbf{B}$ of order $r + r' + 2$ such that for every $w \in \{0, 1\}^n$, $\mathbf{tm}_{r,r',n}(\mathbf{w}_{1,n}) =_{\beta\eta} \mathbf{tt}$ iff TM accepts w. The term $\mathbf{tm}_{r,r',n}(\mathbf{w}_{1,n})$ can be constructed in time polynomial in n.*

Proof. Define types $\mathbf{Tape} := \mathbf{Addr}_{r,n} \to \mathbf{B_3}$ and $\mathbf{Config} := \mathbf{Tape} \times \mathbf{Addr}_{r,n} \times \mathbf{B_q}$. Each $\langle t, \mathbf{k}, \mathbf{i}\rangle : \mathbf{Config}$ represents the configuration with tape content t, head position k and state i. Note that $Order(\mathbf{Config}) = r + 1$.

With the help of $\mathbf{suc}_{r,n}$, $\mathbf{pred}_{r,n}$ and $<_{r,n}$, it is easy to represent the one-step transition of TM by a term $\mathbf{tran} : \mathbf{Config} \to \mathbf{Config}$ of size $O(n^2)$ and order $2r$.

Now the result of the computation after $hyp(r', n)$ steps can be computed by the term $\mathbf{tm}_{r,r',n} := \lambda x.\mathbf{output}(\mathbf{hyp}(r', n)\ \mathbf{tran}\ \mathbf{init}(x)) : \mathbf{B}^n \to \mathbf{B}$ where $\mathbf{init} : \mathbf{B}^n \to \mathbf{Config}$ and $\mathbf{output} : \mathbf{Config} \to \mathbf{B}$ are suitable terms for initialization and output extraction. $\mathbf{hyp}(r', n)$ is considered to be of type $\mathbf{N}_0(\mathbf{Config})$, thus of order $r' + 1 + Order(\mathbf{Config}) = r + r' + 2$. Since $r' \geq r$, this dominates the order $2r$ of $\mathbf{tran}$.

It remains to show that product types can be eliminated from the encoding with at most polynomial size overhead, that will be discussed in the next subsection. ◀

▶ **Corollary 20.**

1. *Both* Bool$(2r + 2)$ *and* RegLang$(2r + 2)$ *are hard for r-*EXPTIME.
2. *Both* Bool$(2r + 3)$ *and* RegLang$(2r + 3)$ *are hard for r-*EXPSPACE.

Proof. Let $r' = r$ for Bool$(2r + 2)$ and $r' = r + 1$ for Bool$(2r + 3)$. It is straightforward to reduce Bool(r) to RegLang(r). ◀

6.2 Elimination of product types

The elimination is based on two isomorphisms:

$$\sigma_1 \times \sigma_2 \to \tau \cong \sigma_1 \to \sigma_2 \to \tau, \qquad \tau \to \sigma_1 \times \sigma_2 \cong (\tau \to \sigma_1) \times (\tau \to \sigma_2).$$

For each type σ, we define a number $b(\sigma)$ (the *breadth*) and a list $\overline{\sigma}$ of length $b(\sigma)$ below. The ith member of the list $\overline{\sigma}$ will be denoted by $(\sigma)_i$.

$$\begin{array}{rcl@{\qquad}rcl} b(o) & := & 1 & \overline{o} & := & o \\ b(\sigma \to \tau) & := & b(\tau) & \overline{\sigma \to \tau} & := & \overline{\sigma} \to (\tau)_1, \ldots, \overline{\sigma} \to (\tau)_m \\ b(\sigma \times \tau) & := & b(\sigma) + b(\tau) & \overline{\sigma \times \tau} & := & \overline{\sigma}, \overline{\tau}, \end{array}$$

where $m = b(\tau)$, and $\overline{\sigma} \to (\tau)_i$ is a shorthand for $(\sigma)_1 \to \cdots (\sigma)_k \to (\tau)_i$ (with $k = b(\sigma)$).

For each term $M : \sigma$, we define a list $\overline{M}$ of length $b(\sigma)$ such that $(M)_i$ (the ith member) is of type $(\sigma)_i$. In the following, we assume that $b(\sigma) = m$ and $b(\tau) = k$.

$$\begin{array}{rcl@{\qquad}rcl} \overline{x^\sigma} & := & x_1^{(\sigma)_1}, \ldots, x_m^{(\sigma)_m} & \overline{\langle M, N\rangle} & := & \overline{M}, \overline{N} \\ \overline{\lambda x^\sigma.M^\tau} & := & \lambda\overline{x^\sigma}.(M)_1, \ldots, \lambda\overline{x^\sigma}.(M)_k & \overline{\pi_1(M^{\sigma\times\tau})} & := & (M)_1, \ldots, (M)_m \\ \overline{M^{\sigma\to\tau}N^\sigma} & := & (M)_1\overline{N}, \ldots, (M)_k\overline{N} & \overline{\pi_2(M^{\sigma\times\tau})} & := & (M)_{m+1}, \ldots, (M)_{m+k}, \end{array}$$

where $x_1, \dots, x_m$ are new variables associated to x, $\lambda\overline{x^\sigma}.(M)_i$ and $(M_i)\overline{N}$ are respectively shorthands for $\lambda x_1 \cdots x_m.(M)_i$ and $(M_i)(N)_1 \cdots (N)_m$.

The translation preserves $\beta\eta$-equality:

▶ **Theorem 21.** *If $M : \sigma$ and $M =_{\beta\eta} N$, then $(M)_i =_{\beta\eta} (N)_i$ for every $1 \leq i \leq b(\sigma)$.*

Therefore, when applied to $\mathbf{tm}_{r,r',n}(\mathbf{u}_{1,n}) : \mathbf{B}$, the translated term returns the correct output. However, we have to be careful for size. Indeed, if $\mathbf{n} : \mathbf{N}(\sigma)$, we have $|(\mathbf{n})_i| = O(b(\sigma)^n)$. To avoid such an unexpected explosion, we define the *depth* $d(M)$ of each term M as follows:

$$\begin{array}{rclrclrcl} d(x) & := & 1 & d(\lambda x.M) & := & d(M) & d(\langle M, N\rangle) & := & \max(d(M), d(N)) \\ & & & d(MN) & := & \max(d(M), d(N)+1) & d(\pi_i M) & := & d(M). \end{array}$$

We can now estimate the size after translation.

▶ **Lemma 22.** *Let $M : \sigma$, $m = \max\{b(\tau) : \tau \in \mathit{Type}(M)\}$ and $d = d(M)$. Then $|(M)_i| \leq m^d \cdot |M|$ for every $1 \leq i \leq b(\sigma)$.*

For instance, $d(\mathbf{n}) = O(n)$ and this is the reason of the exponential explosion. It is important for our purpose that the order of association matters for iterated multiplications $\mathbf{2} \circ \cdots \circ \mathbf{2}$ (n times). Depending on whether we associate them to the right or to the left, the depth changes significantly:

$$d(\mathbf{2} \circ (\cdots (\mathbf{2} \circ (\mathbf{2} \circ \mathbf{2})))) = O(n), \qquad d((((\mathbf{2} \circ \mathbf{2}) \circ \mathbf{2}) \cdots) \circ \mathbf{2}) = O(1).$$

Hence if we choose the second one for the definition of $\mathbf{hyp}(r, n)$ that appears in $\mathbf{tm}_{r,r',n}$, then the depth stays constant (as far as r, r' are fixed), so that we obtain a term of polynomial size by translation. Therefore, Theorem 19 holds without product types.

7 Conclusion

We have studied the computational complexity of evaluating simply typed lambda terms of bounded order. As observed by [8, 9, 14], efficiency is achieved by mixing β reduction and semantic evaluation. In comparison with these previous works, the crux of our work lies in the use of $\mathbf{ScottL_!}$ rather than **Set**. Indeed, the former has led to an intersection type system, and thus provided a natural alternating algorithm. Moreover, it is flexible enough to incorporate Krivine's machinery and restriction to thin types. All these ideas combine into an optimal algorithm that comforms to the completeness results for the problems Bool(r) and RegLang(r), although there still remains a gap for Term(r).

It should be mentioned that semantic evaluation is also quite effective in the setting of infinitary lambda calculus [1]. Incidentally, [11] refines [1] by replacing semantic evaluation in **Set** by intersection typing. This is strictly parallel to our shift from **Set** to $\mathbf{ScottL_!}$.

We believe that this way of using semantics for computation should be further developed, so let us conclude with a slogan: *better semantics, faster computation.*

Acknowledgements We would like to thank Masahito Hasegawa, Naohiko Hoshino and Shin'ya Katsumata for pointers to the literature, and anonymous referees for a lot of useful comments. This work is supported by JSPS KAKENHI 21700041.

References

1 Klaus Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.

2 Arnold Beckmann. Exact bounds for lengths of reductions in typed lambda-calculus. *J. Symb. Log.*, 66(3):1277–1285, 2001.

3 Richard Blute and Philip Scott. Category theory for linear logicians. In *Linear Logic in Computer Science, LMS 316*. Cambridge University Press, 2004.

4 Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *Mathematical Structures in Computer Science*, to appear.

5 Lorenzo Tortora de Falco. Obsessional experiments for linear logic proof-nets. *Mathematical Structures in Computer Science*, 13(6):799–855, 2003.

6 Thomas Ehrhard. The scott model of linear logic is the extensional collapse of its relational model. *Theor. Comput. Sci.*, to appear.

7 Harvey Friedman. Equality between functionals. In *Proceedings of Logic Colloquium'73, LNM 453*, pages 22–37, 1975.

8 Andreas Goerdt and Helmut Seidl. Characterizing complexity classes by higher type primitive recursive definitions, Part II. In *Proc. 6th IMYCS, LNCS 464*, pages 148–158, 1990.

9 Gerd G. Hillebrand and Paris C. Kanellakis. On the expressive power of simply typed and let-polymorphic lambda calculi. In *Proceedings of 11th LICS*, pages 253–263, 1996.

10 Michael Huth. Linear domains and linear maps. In *Proceedings of 9th MFPS, LNCS 802*, pages 438–453, 1993.

11 Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of 36th POPL*, pages 416–428, 2009.

12 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of 24th LICS*, pages 179–188, 2009.

13 Lars Kristiansen. Complexity-theoretic hierarchies induced by fragments of Gödel's T. *Theory Comput. Syst.*, 43(3-4):516–541, 2008.

14 Lars Kristiansen and Paul J. Voda. Programming languages capturing complexity classes. *Nord. J. Comput.*, 12(2):89–115, 2005.

15 Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.

16 Harry G. Mairson. A simple proof of a theorem of Statman. *Theor. Comput. Sci.*, 103(2):387–394, 1992.

17 Paul-Andre Mellies. Categorical semantics of linear logic. In *Interactive models of computation and program behaviour, Panoramas et Syntheses 27*. Soc. Math. de France, 2009.

18 Sylvain Salvati. On the membership problem for non-linear abstract categorial grammars. *Journal of Logic, Language and Information*, 19(2):163–183, 2010.

19 Aleksy Schubert. The complexity of β-reduction in low orders. In *Proceedings of 5th TLCA, LNCS 2044*, pages 400–414, 2001.

20 Richard Statman. The typed lambda-calculus is not elementary recursive. *Theor. Comput. Sci.*, 9:73–81, 1979.

21 Steffen van Bakel. Intersection type assignment systems. *Theor. Comput. Sci.*, 151(2):385–435, 1995.

22 Steffen van Bakel. Strict intersection types for the lambda calculus. *ACM Comput. Surv.*, 43(3):20, 2011.

23 Glynn Winskel. A linear metalanguage for concurrency. In *Proceedings of 7th AMAST, LNCS 1548*, pages 42–58, 1998.

24 Glynn Winskel. Linearity and non linearity in distributed computation. In *Linear Logic in Computer Science, LMS 316*. Cambridge University Press, 2004.

On the Formalization of Termination Techniques based on Multiset Orderings*

René Thiemann[1], Guillaume Allais[2], and Julian Nagele[1]

1 University of Innsbruck, 6020 Innsbruck, Austria
2 University of Strathclyde, Glasgow G1 1XQ, Scotland

Abstract

Multiset orderings are a key ingredient in certain termination techniques like the recursive path ordering and a variant of size-change termination. In order to integrate these techniques in a certifier for termination proofs, we have added them to the *Isabelle Formalization of Rewriting*. To this end, it was required to extend the existing formalization on multiset orderings towards a generalized multiset ordering. Afterwards, the soundness proofs of both techniques have been established, although only after fixing some definitions.

Concerning efficiency, it is known that the search for suitable parameters for both techniques is NP-hard. We show that checking the correct application of the techniques—where all parameters are provided—is also NP-hard, since the problem of deciding the generalized multiset ordering is NP-hard.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases formalization, term rewriting, termination, orderings

Digital Object Identifier 10.4230/LIPIcs.RTA.2012.339

Category Regular Research Paper

1 Introduction

The multiset ordering has been invented in the '70s to prove termination of programs [10]. It is an ingredient for important termination techniques like the multiset path ordering (MPO) [8], the recursive path ordering (RPO) [9], or recently [3, 6] it has been used in combination with the size-change principle of [19], in the form of SCNP reduction pairs.

The original version of the multiset ordering w.r.t. some base ordering $\succ$ can be defined as $M \succ_{ms} N$ iff it is possible to obtain N from M by replacing at least one element of M by several strictly smaller elements in N.

In other papers—like [3, 6, 22]—a generalization of the multiset ordering is used (denoted by $\succ_{gms}$). To define $\succ_{gms}$ one assumes that in addition to $\succ$ there is a compatible non-strict ordering $\succsim$. Then $\succ_{gms}$ is like $\succ_{ms}$, but in the multiset comparison it is additionally allowed to replace each element by a smaller one (w.r.t. $\succsim$). To illustrate the difference between $\succ_{ms}$ and $\succ_{gms}$, we take $\succ$ and $\succsim$ as the standard orderings on polynomials over the naturals. Then $\succ_{gms}$ is strictly more powerful than $\succ_{ms}$: for example, $\{\{x+1, 2y\}\} \succ_{gms} \{\{y, x\}\}$ since $x + 1 \succ x$ and $2y \succsim y$, but $\{\{x + 1, 2y\}\} \succ_{ms} \{\{y, x\}\}$ does not hold, since $2y \neq y$ and also $2y \not\succ y$ as y can be instantiated by 0.

* This research is supported by FWF (Austrian Science Fund) project P22767.

Note that $\succ_{gms}$ is indeed used in powerful termination tools like AProVE [11]. Hence, for the certification of termination proofs which make use of SCNP reduction pairs or RPOs which are defined via $\succ_{gms}$, we need a formalization of this multiset ordering.

However, in the literature and in formalizations, often only $\succ_{ms}$ is considered. And even those papers that use $\succ_{gms}$ only shortly list the differences between $\succ_{ms}$ and $\succ_{gms}$—if at all—and afterwards just assume that both multiset orderings have similar properties.

To change this situation, as one new contribution of this paper, we give the first formalization of $\succ_{gms}$, and could indeed show that $\succ_{ms}$ and $\succ_{gms}$ behave quite similar. However, we also found one essential difference between both orderings. It is well known that deciding $\succ_{ms}$ is easy, one just removes identical elements and afterwards has to find for each element in the one set a larger element in the other. In contrast, we detected and proved that deciding $\succ_{gms}$ is an NP-complete problem.

Note that the original definition of RPO misses a feature that is present in other orderings like polynomial interpretations where it is possible to compare variables against a least element: $x \succsim 0$. This feature was already integrated in other orderings like KBO [17], and it can also be integrated into RPO where one allows to compare $x \succsim c$ if c is a constant of least precedence. For example, the internal definition of RPO in AProVE is the one of [22]—which is based on $\succ_{gms}$—with the additional inference rule of "$x \succsim c$". As a second new contribution we formalized this RPO variant and fixed its definition, as it turned out that it is not stable. Moreover, we show that the change from $\succ_{ms}$ to $\succ_{gms}$ increases the complexity of RPO: From [18] it is known that deciding whether two terms are in relation w.r.t. a given RPO or MPO is in P. However, if one uses the definitions of MPO and RPO in this paper which are based on $\succ_{gms}$, then the same decision problem becomes NP-complete.

As third new contribution we also give the first formalization of SCNP reduction pairs where we could establish the main soundness theorem, although several definitions had to be fixed. Again, we have shown that the usage of $\succ_{gms}$ makes certification for SCNP reduction pairs an NP-complete problem. As a consequence, the search for suitable SCNP reduction pairs and the problem whether two terms are in relation for a given SCNP reduction pair belong to the same complexity class, as they are both NP-complete.

All our formalizations are using the proof assistant Isabelle/HOL [21]. They are available within the IsaFoR library (Isabelle Formalization of Rewriting, [24]). The new parts of the corresponding proof checker CeTA—which is just obtained by applying the code generator of Isabelle [13] on IsaFoR—have been tested on the examples of the experiments performed in [22]. After fixing some output bugs, eventually all proofs could be certified. Both IsaFoR and CeTA are freely available at:

`http://cl-informatik.uibk.ac.at/software/ceta`

The paper is structured as follows. We give preliminaries and the exact definitions for $\succ_{ms}$ and $\succ_{gms}$ in Sec. 2. Afterwards we discuss the formalization of $\succ_{gms}$ in Sec. 3. Different variants of RPO are discussed in Sec. 4. Here, we also show that certifying constraints for the RPO variant used in AProVE is NP-complete. In Sec. 5 we discuss the formalization of SCNP reduction pairs. Finally, in Sec. 6 we elaborate on our certified algorithms for checking whether two terms are in relation w.r.t. RPO or the orderings from an SCNP reduction pair, and we report on our experimental results.

2 Preliminaries

We refer to [2] for basic notions and notations of rewriting. A *signature* is a set of symbols $\mathcal{F} = \{f, g, F, G, \dots\}$, each associated with an *arity*. We write $\mathcal{T}(\mathcal{F}, \mathcal{V})$ for the set of terms over signature $\mathcal{F}$ and set of variables $\mathcal{V}$. We write $\mathcal{V}(t)$ for the set of variables occurring in t. A relation $\succ$ on $\mathcal{T}(\mathcal{F},\mathcal{V})$ is *stable* iff it is closed under substitutions, and it is *monotone*, iff it is closed under contexts. A *term rewrite system* (TRS) is a set of *rules* $\ell \to r$. The rewrite relation $\to_\mathcal{R}$ of a TRS $\mathcal{R}$ is the smallest stable and monotone relation containing $\mathcal{R}$.

We write *Id* for the identity relation, and for each relation $\succ$, let $\succeq$ be its reflexive closure.

▶ **Definition 2.1** (Ordering pair, reduction pair)**.** The pair $(\succsim, \succ)$ is an *ordering pair* (over carrier A) iff $\succsim$ is a quasi-ordering over A, $\succ$ is a transitive and well-founded relation over A, and $\succ$ and $\succsim$ are compatible, i.e., $\succ \circ \succsim \subseteq \succ$ and $\succsim \circ \succ \subseteq \succ$.

The pair $(\succsim, \succ)$ is a *non-monotone reduction pair* iff $(\succsim, \succ)$ is an ordering pair over $\mathcal{T}(\mathcal{F},\mathcal{V})$ where both $\succ$ and $\succsim$ are stable. If additionally $\succsim$ is monotone, then $(\succsim, \succ)$ is a *reduction pair*, and if both $\succ$ and $\succsim$ are monotone, then $(\succsim, \succ)$ is a *monotone reduction pair*.

Throughout this paper we only consider finite multisets $\{\{x_1, \dots, x_n\}\}$ and we write $\mathfrak{P}(A)$ for the set of all multisets with elements from A. For every function $f : A \to A$ and every multiset $M \in \mathfrak{P}(A)$ we define the image of f on M as $f[M] = \{\{f(x) \mid x \in M\}\}$.

▶ **Definition 2.2** (Multiset orderings)**.** Let $\succ$ and $\succsim$ be relations over A. We define the *multiset ordering* ($\succ_{ms}$), the *generalized multiset ordering* ($\succ_{gms}$), and the corresponding non-strict ordering ($\succsim_{gms}$) over $\mathfrak{P}(A)$ as follows: $M_1 \succ_{ms} / \succ_{gms} / \succsim_{gms} M_2$ iff there are S_i and E_i such that $M_1 = S_1 \cup E_1$, $M_2 = S_2 \cup E_2$, and

- conditions 1, 2, and 4 are satisfied: $M \succ_{ms} N$
- conditions 1, 3, and 4 are satisfied: $M \succ_{gms} N$
- conditions 1 and 3 are satisfied: $M \succsim_{gms} N$

where conditions 1–4 are defined by:

1. for each $y \in S_2$ there is some $x \in S_1$ with $x \succ y$
2. $E_1 = E_2$
3. $E_1 = \{\{x_1, \dots, x_n\}\}$, $E_2 = \{\{y_1, \dots, y_n\}\}$, and $x_i \succsim y_i$ for all $1 \le i \le n$
4. $S_1 \neq \emptyset$

Whenever M_i is split into $S_i \cup E_i$ we call S_i the *strict part* and E_i the *non-strict part*.

Note that $\succ_{gms}$ indeed generalizes $\succ_{ms}$, since $\succ_{gms} = \succ_{ms}$ if $\succsim = Id$.

3 Formalization of the Generalized Multiset Ordering

Replacing condition 2 by 3 makes the formalization of $\succ_{gms}$ a bit more involved than the one of $\succ_{ms}$: the simple operation of (multiset) equality $E_1 = E_2$ is replaced by demanding that both E_1 and E_2 can be enumerated in such a way that $x_i \succsim y_i$ for all $1 \le i \le n$. Note that instead of enumerations one can equivalently demand that there is a bijection $f : E_1 \to E_2$ such that for all $x \in E_1$ we have $x \succsim f(x)$.

There is one main advantage of formalizing condition 3 via enumerations instead of bijections. It will be rather easy to develop an algorithm deciding $\succ_{gms}$. However, using enumerations we also observe a drawback: the proof that $\succ_{gms}$ and $\succsim_{gms}$ are compatible and transitive orderings will be harder than using bijections since composing bijections is easier than combining enumerations.

The formalization of the following (expected) properties for $\succ_{gms}$ and $\succsim_{gms}$ has been rather simple.

▶ **Lemma 3.1.** *$\succ_{gms}$ and $\succsim_{gms}$ have the following properties:*

1. *the empty set is the unique minimum of $\succ_{gms}$ and $\succsim_{gms}$*
2. *if $\succsim$ is reflexive then so is $\succsim_{gms}$*

3. *if $\succ$ is irreflexive and compatible with $\succsim$ then $\succ_{gms}$ is irreflexive*
4. *if $\succ$ and $\succsim$ are closed under an operation op then $\succ_{gms}$ and $\succsim_{gms}$ are closed under $op[\cdot]$.*

In contrast, the formalization of transitivity and compatibility was more tedious.

▶ **Lemma 3.2.** *If $\succ$ and $\succsim$ are compatible and transitive then so are $\succ_{gms}$ and $\succsim_{gms}$.*

Proof. For the sake of simplicity, we will work here with the definition of $(\succ_{gms}, \succsim_{gms})$ using bijections rather than enumerations: all technical details when using the representation with enumerations are available in IsaFoR, theory *Multiset-Extension*. Given that this lemma states four results that are pretty similar, we will only prove transitivity of $\succsim_{gms}$ and let the reader see how the proof could be slightly modified in the other cases.

Let M, N and P be three multisets such that $M \succsim_{gms} N$ (1) and $N \succsim_{gms} P$ (2). From (1) we get the partitions $M = M' \cup E$ and $N = N' \cup F$ and the bijection $f_{MN} : E \to F$. From (2) we get the partitions $N = N'' \cup F'$ and $P = P' \cup G$ and the bijection $f_{NP} : F' \to G$. We define the multiset I as $I = F \cap F'$ and claim that the partitions $M = (M \setminus f_{MN}^{-1}[I]) \cup f_{MN}^{-1}[I]$ and $P = (P \setminus f_{NP}[I]) \cup f_{NP}[I]$ and the bijection $f_{NP} \circ f_{MN}$ are the ones needed to prove $M \succsim_{gms} P$.

If $p \in P \setminus f_{NP}[I]$ then we have to find some $m \in M \setminus m \notin f_{MN}^{-1}[I]$ satisfying $m \succ p$. We distinguish two cases. In the first case, $p \in P'$ and thus there is an $n \in N''$ such that $n \succ p$ and $n \notin I$. If $n \in N'$ then by (1) there is some $m \in M'$ with $m \succ n$ and hence, $m \succ p$ by transitivity of $\succ$. Moreover, since $M' \subseteq M \setminus f_{MN}^{-1}[I]$ we found the desired element m. Otherwise, $n \in F$ and hence, for $m = f_{MN}^{-1}(n)$ we know $m \succsim n$ using (1). By compatibility, also $m \succ p$. Again, $m \in M \setminus f_{MN}^{-1}[I]$ since $n \notin I$. In the second case, $p \in G$ and thus for $n = f_{NP}^{-1}(p)$ we conclude $n \in F'' \setminus I$ and $n \succsim p$, and hence $n \notin F$. Thus, there is an element $m \in M'$ which satisfies $m \succ n$. By compatibility we again achieve $m \succ p$.

If $p \in f_{NP}[I]$, we have an element $n \in F'$ such that $n \succsim p$; n is also in F and therefore we have an element $m \in f_{MN}^{-1}[I]$ such that $m \succsim n \succsim p$ and hence, $m \succsim p$. ◀

Here lies the main difference to the formalization of $\succ_{ms}$ in [16]. While just transitivity needs to be shown for $\succ_{ms}$, we had to show transitivity of both $\succ_{gms}$ and $\succsim_{gms}$ as well as compatibility from both sides. Moreover the formalized proofs of these facts get more complicated since we cannot simply use bijections, set intersections, and differences, but have to deal with enumerations.

After having established Lem. 3.1 and Lem. 3.2 it remains to prove the most interesting and also most complicated property of $\succ_{gms}$, namely strong normalization. In the remainder of this section we assume that $\succ$ and $\succsim$ are compatible and transitive, and that $\succ$ is strongly normalizing. Our proof is closely related to the one for $\succ_{ms}$ [20] which is due to Buchholz: we first introduce a more atomic relation $\succ_{gms\text{-}step}$ which has $\succ_{gms}$ as its transitive closure. Then it suffices to prove that $\succ_{gms\text{-}step}$ is well-founded.

▶ **Definition 3.3.** We define $\succ_{gms\text{-}step}$ as $M \succ_{gms\text{-}step} N$ iff $M \succ_{gms} N$ where in the split $M = S \cup E$ the size of S is exactly one.

▶ **Lemma 3.4.** *$\succ_{gms}$ is the transitive closure of $\succ_{gms\text{-}step}$.*

For strong normalization we perform an accessibility-style proof, i.e., we define $\mathcal{A}$ as the set of strongly normalizing elements w.r.t. $\succ_{gms\text{-}step}$ and show that $\mathcal{A}$ contains all multisets. Note that to show $M \in \mathcal{A}$ it suffices to prove strong normalization for all N with $M \succ_{gms\text{-}step} N$. Moreover, since $\succ_{gms\text{-}step}$ is strongly normalizing on $\mathcal{A}$, one can use the following induction principle to prove some property P for all elements in $\mathcal{A}$.

$$(\forall M.(\forall N \in \mathcal{A}.M \succ_{gms\text{-}step} N \to P(N)) \to P(M)) \to (\forall M \in \mathcal{A}.P(M)) \qquad (\star)$$

We will later on apply this induction scheme using the first of the following two predicates:

- $P(x)$ is defined as $\forall M. M \in \mathcal{A} \to M \cup \{\{x\}\} \in \mathcal{A}$
- $Q(M, x)$ is defined as $\forall b. x \succsim b \to M \cup \{\{b\}\} \in \mathcal{A}$

For showing that all multisets belong to $\mathcal{A}$, we will require the following technical lemma.

▶ **Lemma 3.5.** *For all multisets $M \in \mathcal{A}$ and for all elements a, if $\forall N. M \succ_{gms\text{-}step} N \to Q(N, a)$ and $\forall b. a \succ b \to P(b)$ then $Q(M, a)$ holds.*

Proof. To prove $Q(M, a)$, let b be an element such that $a \succsim b$ where we have to show $M \cup \{\{b\}\} \in \mathcal{A}$. To prove the latter, we consider an arbitrary N with $M \cup \{\{b\}\} \succ_{gms\text{-}step} N$ and have to show $N \in \mathcal{A}$. From $M \cup \{\{b\}\} \succ_{gms\text{-}step} N$ we obtain suitable $m_1, \ldots, m_k$, $n_1, \ldots, n_k$, m, and N' to perform the splits $M \cup \{\{b\}\} = \{\{m\}\} \cup \{\{m_1, \ldots, m_k\}\}$ and $N = N' \cup \{\{n_1, \ldots, n_k\}\}$. We distinguish two cases: either b is part of $\{\{m_1, \ldots, m_k\}\}$ or equal to m.

If $b = m_i$ for some i then $M \succ_{gms\text{-}step} N \setminus \{\{n_i\}\}$. Thanks to the first hypothesis and the fact that $a \succsim n_i$ (because $a \succsim b = m_i \succsim n_i$), we can conclude that $N \in \mathcal{A}$.

If $b = m$ then one can prove that $\{\{n_1, \ldots, n_k\}\} \in \mathcal{A}$ using the fact that $\{\{m_1, \ldots, m_k\}\} = M \in \mathcal{A}$ and $\forall i. m_i \succsim n_i$. By induction on the size of N' and thanks to the second hypothesis and the fact that for any $p \in N'$, $a \succsim b \succ p$, we deduce that $\{\{n_1, \ldots, n_k\}\} \cup N' \in \mathcal{A}$ which concludes the proof. ◀

▶ **Lemma 3.6.** $\forall M. M \in \mathcal{A}$.

Proof. The proof of the lemma is done by induction on the size of the multiset M.

If M is the empty multiset, then obviously, it is strongly normalizing (hence in $\mathcal{A}$).

Otherwise we have to prove $\forall a. \forall M \in \mathcal{A}. M \cup \{\{a\}\} \in \mathcal{A}$ which is the same as $\forall a. P(a)$. We perform a well-founded induction on a (w.r.t. $\succ$) using the property P and are left to prove $P(a)$ assuming that $\forall b. a \succ b \to P(b)$ holds. We pick a multiset $M \in \mathcal{A}$ and perform an induction on M using ($\star$) to prove the property $Q(M, a)$ (which entails $P(a)$ because $\succsim$ is reflexive): for any $M \in \mathcal{A}$ we have to prove $Q(M, a)$ given the induction hypothesis $\forall N. M \succ_{gms\text{-}step} N \to Q(N, a)$. But this result is a trivial application of Lem. 3.5 using the two induction hypotheses that we just generated. ◀

Using Lem. 3.4 and Lem. 3.6, strong normalization of $\succ_{gms}$ immediately follows.

▶ **Theorem 3.7.** *$\succ_{gms}$ is strongly normalizing.*

4 Multiset and Recursive Path Ordering

In this section we study variations of MPO and RPO. To this end, throughout this section we assume that $\geqslant$ is some precedence for the signature $\mathcal{F}$. We write $> \, = \, \geqslant \setminus \leqslant$ for the strict part of the precedence, and $\approx \, = \, \geqslant \cap \leqslant$ for its equivalence relation.

A standard version of MPO allowing quasi-precedences can be defined by the following inference rules.

$$\frac{s_i \succeq^{mpo} t}{f(\vec{s}) \succ^{mpo} t} \qquad \frac{\{\{\vec{s}\}\} \succ^{mpo}_{ms} \{\{\vec{t}\}\}}{f(\vec{s}) \succ^{mpo} g(\vec{t})}\, f \approx g \qquad \frac{f(\vec{s}) \succ^{mpo} t_i \text{ for all } i}{f(\vec{s}) \succ^{mpo} g(\vec{t})}\, f > g$$

For example for the precedence where $\mathsf{g} \approx \mathsf{h}$ we conclude that $\mathsf{f}(y, \mathsf{s}(x)) \succ^{mpo} \mathsf{f}(x, y)$ and $\mathsf{g}(\mathsf{s}(x)) \succ^{mpo} \mathsf{h}(x)$, but $\mathsf{f}(\mathsf{g}(y), \mathsf{s}(x)) \not\succ^{mpo} \mathsf{f}(x, \mathsf{h}(y))$ since $\{\{\mathsf{g}(y), \mathsf{s}(x)\}\} \not\succ^{mpo}_{ms} \{\{x, \mathsf{h}(y)\}\}$ as $\mathsf{g}(y) \not\succeq^{mpo} \mathsf{h}(y)$.

To increase the power of MPO, the following inference rules are sometimes used which generalize MPO by defining two orderings $\succ^{gmpo}$ and $\succsim^{gmpo}$ where the multiset extension is done via $\succ_{gms}$. This variant of MPO is the one that is internally used within AProVE, and if one removes the last inference rule ($x \succsim^{gmpo} c$), then it is equivalent to the MPO definition of [22].

$$(1)\ \frac{s_i \succsim^{gmpo} t}{f(\vec{s}) \succ^{gmpo} t} \qquad (2)\ \frac{\{\{\vec{s}\}\} \succ^{gmpo}_{gms} \{\{\vec{t}\}\}}{f(\vec{s}) \succ^{gmpo} g(\vec{t})} f \approx g \qquad (3)\ \frac{f(\vec{s}) \succ^{gmpo} t_i \text{ for all } i}{f(\vec{s}) \succ^{gmpo} g(\vec{t})} f > g$$

$$(4)\ \frac{s \succeq^{gmpo} t}{s \succsim^{gmpo} t} \qquad (5)\ \frac{\{\{\vec{s}\}\} \succsim^{gmpo}_{gms} \{\{\vec{t}\}\}}{f(\vec{s}) \succsim^{gmpo} g(\vec{t})} f \approx g \qquad (6)\ \frac{}{x \succsim^{gmpo} c} \text{if } \forall f \in \mathcal{F} : f \geqslant c$$

Hence, there are two differences between $\succeq^{gmpo}$ (the reflexive closure of $\succ^{gmpo}$) and $\succsim^{gmpo}$: only in $\succsim^{gmpo}$ one can compare multisets using the non-strict multiset ordering, and one can compare variables against constants of least precedence. This latter feature is similar to polynomial orderings where $x \geqslant 0$, and it was also added to the Knuth-Bendix-Ordering [17].

The increase of power of $\succ^{gmpo}$ in comparison to $\succ^{mpo}$ is due to both new features in the non-strict relation. For example, using the same precedence as before, $\mathsf{f}(\mathsf{g}(y), \mathsf{s}(x)) \succ^{gmpo} \mathsf{f}(x, \mathsf{h}(y))$ as $\mathsf{g}(y) \succsim^{gmpo} \mathsf{h}(y)$ and $\mathsf{s}(x) \succ^{gmpo} x$. Moreover, $\mathsf{f}(\mathsf{f}(y,z), \mathsf{s}(x)) \succ^{gmpo} \mathsf{f}(x, \mathsf{f}(\mathsf{a}, z))$ if a has least precedence, where this decrease is only possible due to the comparison $y \succsim^{gmpo} \mathsf{a}$.

Note that $\succsim^{gmpo}$ is a strict superset of the equivalence relation where equality is defined modulo $\approx$ and modulo permutations. For this inclusion, indeed all three inference rules (4-6) of $\succsim^{gmpo}$ are essential. With (4) and (5) we completely subsume the equivalence relation, and (6) exceeds the equivalence relation. However, then also (5) adds additional power by using $\succsim_{gms}$ instead of multiset equality w.r.t. the equivalence relation. As an example, consider $\mathsf{g}(x) \succsim^{gmpo} \mathsf{g}(\mathsf{a})$.

Finally note that our definition does not require any explicit definition of an equivalence relation, one can just use $\succsim^{gmpo} \cap \precsim^{gmpo}$. This is in contrast to the RPO in related formalizations of CoLoR [5] and Coccinelle [7]. In Coccinelle first an equivalence relation for RPO is defined explicitly, before defining the strict ordering,[1] and the RPO of CoLoR currently just supports syntactic equality.[2]

The reason that AProVE uses $\succ^{gmpo}$ for its MPO implementation is easily understood, as $\succ^{gmpo}$ is more powerful than $\succ^{mpo}$, and the SAT/SMT-encodings of $\succ^{mpo}$ and $\succ^{gmpo}$ to find a suitable precedence are quite similar, so there is not much overhead. Hence, in order to be able to certify AProVE's MPO proofs—which then also allows to certify weaker variants of MPO—we have formally proved that $(\succsim^{gmpo}, \succ^{gmpo})$ is a monotone reduction pair. Concerning strong normalization of $\succ^{gmpo}$, we did not use Kruskal's tree theorem, but we performed a proof similar to the strong normalization proof of the (higher-order) recursive path ordering as in [5, 7, 15, 16] (which is based on reducibility predicates of Tait and Girard.) By following this proof and by using the results of Sec. 3, it was an easy—but tedious—task to formalize the following main result. Here—as for the generalized multiset ordering—the transitivity proof became more complex as one has to prove transitivity and compatibility of $\succsim^{gmpo}$ and $\succ^{gmpo}$ at the same time, i.e., within one large inductive proof.

► **Theorem 4.1.** *The pair $(\succsim^{gmpo}, \succ^{gmpo})$ is a monotone reduction pair.*

Whereas Thm. 4.1 was to be expected—$\succ^{gmpo}$ is just an extension of $\succ^{mpo}$, and it is well known that $(\succeq^{mpo}, \succ^{mpo})$ is a monotone reduction pair—we detected a major difference

[1] See `http://www.lri.fr/~contejea/Coccinelle/doc/term_orderings.rpo.html`.
[2] See `http://color.inria.fr/doc/CoLoR.RPO.VRPO_Type.html`.

when trying to certify existing proofs where one has to compute for a given precedence whether two terms are in relation. This problem is in P for $\succ^{mpo}$ but it turns out to be NP-complete for $\succ^{gmpo}$.

▶ **Theorem 4.2.** *Let there be some fixed precedence. The problem of deciding $\ell \succ^{gmpo} r$ for two terms ℓ and r is NP-complete.*

Proof. Membership in NP is easily proved. Since the size of every proof-tree for $\ell \succ^{mpo} r$ is bounded by $2 \cdot |\ell| \cdot |r|$, one can just guess how the inference rules for $\succ^{mpo}$ have to be applied; and for the multiset comparisons one can also just guess the splitting.

To show NP-hardness we perform a reduction from SAT. So let ϕ be some Boolean formula over variables $\{x_1, \dots, x_n\}$ represented as a set of clauses $\{C_1, \dots, C_m\}$ where each clause is a set of literals, and each literal l is variable x_i or a negated variable $\overline{x_i}$. W.l.o.g. we assume $n \geq 2$.

In the following we will construct one constraint $\ell \succ^{gmpo} r$ for terms $\ell, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ where $\mathcal{F} = \{\mathsf{a}, \mathsf{f}, \mathsf{g}, \mathsf{h}\}$ and $\mathcal{V} = \{x_1, \dots, x_n, y_1, \dots, y_m\}$. To this end, we define $s(l, C_j) = y_j$, if $l \in C_j$, and $s(l, C_j) = \mathsf{a}$, otherwise. Moreover, $t_x^+ = \mathsf{f}(x, s(x, C_1), \dots, s(x, C_m))$, $t_x^- = \mathsf{f}(x, s(\overline{x}, C_1), \dots, s(\overline{x}, C_m))$, $t_x = \mathsf{f}(x, \mathsf{a}, \dots, \mathsf{a})$. We define $L = \{\{t_{x_1}^+, t_{x_1}^-, \dots, t_{x_n}^+, t_{x_n}^-\}\}$ and $R = \{\{t_{x_1}, \dots, t_{x_n}, y_1, \dots, y_m\}\}$. Finally, we define $\ell = \mathsf{g}(L)$ and $r = \mathsf{h}(R)$—where here we abuse notation and interpret L and R as lists of terms.

We prove that ϕ is satisfiable iff $\ell \succ^{gmpo} r$ for the precedence where $\mathsf{a} \approx \mathsf{f} \approx \mathsf{g} \approx \mathsf{h}$. For this precedence, $\ell \succ^{gmpo} r$ iff $L \succ_{gms}^{gmpo} R$ since there is only one inference rule that can successfully be applied. The reason is that $\mathsf{f} \not\succ \mathsf{g}$ and for each term $t_{x_i}^\pm$ of L we have $t_{x_i}^\pm \not\succsim^{gmpo} r$ where $t_{x_i}^\pm$ represents one of the terms $t_{x_i}^+$ or $t_{x_i}^-$. To see the latter, assume $t_{x_i}^\pm \succsim^{gmpo} r$ would hold. Then $\{x_i, y_1, \dots, y_m\} \supseteq \mathcal{V}(t_{x_i}^\pm) \supseteq \mathcal{V}(r) \supseteq \{x_1, \dots, x_n\}$ being a contradiction to $n \geq 2$.

To examine whether $L \succ_{gms}^{gmpo} R$ can hold, let us consider an arbitrary splitting of R into a strict part S' and a non-strict part E'. Notice that $t_x^+ \succsim^{gmpo} t_x$ and $t_x^- \succsim^{gmpo} t_x$, but neither $t_x^+ \succ^{gmpo} t_x$ nor $t_x^- \succ^{gmpo} t_x$: the reason is that for each l and C_j we get $s(l, C_j) \succsim^{gmpo} \mathsf{a}$ but not $s(l, C_j) \succ^{gmpo} \mathsf{a}$. Hence, each t_x of R must be contained in E'. As moreover, $t_{x_i}^\pm \succ^{gmpo} y_j$ iff $t_{x_i}^\pm \succsim^{gmpo} y_j$ we can w.l.o.g. assume that each $y_j \in S'$. In total, if $L \succ_{gms}^{gmpo} R$, then R must be split into $S' \cup E'$ where $S' = \{\{y_1, \dots, y_m\}\}$ and $E' = \{\{t_{x_1}, \dots, t_{x_n}\}\}$ and L must be split into $S \cup E$ such that all conditions of $\succ_{gms}^{gmpo}$ are satisfied.

At this point we consider both directions to show that ϕ is satisfiable iff $L \succ_{gms}^{gmpo} R$.

First assume that ϕ is satisfiable, so let α be some satisfying assignment. Then we choose $S = \{\{t_x^+ \mid \alpha(x) = \top\}\} \cup \{\{t_x^- \mid \alpha(x) = \bot\}\}$ and $E = L \setminus S$. Notice that for each x exactly one of t_x^+ and t_x^- is in S, and the other is in E. Hence, for each $t_x \in E'$ there is a corresponding $t_x^\pm \in E$ with $t_x^\pm \succsim^{gmpo} t_x$. Next, we have to find for each $y_j \in S'$ some term in S which is larger than y_j. To this end, notice that α is a satisfying assignment, thus there is some literal x_i or $\overline{x_i}$ in C_j which evaluates to true. If $x_i \in C_j$ then $\alpha(x_i) = \top$ and hence, $t_{x_i}^+ \in S$ where $t_{x_i}^+ = \mathsf{f}(\dots, y_j, \dots) \succ^{gmpo} y_j$. Otherwise, $\overline{x_i} \in C_j$ and $\alpha(x_i) = \bot$ and hence, $t_{x_i}^- \in S$ where $t_{x_i}^- = \mathsf{f}(\dots, y_j, \dots) \succ^{gmpo} y_j$. Thus, in both cases there is some term in S being larger than y_j. Moreover, $S \neq \emptyset$ since $|S| = n \geq 2$.

For the other direction assume that S and E could be found such that $L = S \cup E$ and all conditions of $\succ_{gms}^{gmpo}$ are satisfied. Hence for each $t_{x_i} \in E'$ there is some term $t \in E$ satisfying $t \succsim^{gmpo} t_{x_i}$. Then, t can only be $t_{x_i}^+$ or $t_{x_i}^-$ since $x_i \in \mathcal{V}(t_x)$ and each other term $t_{x_j}^\pm$ with $i \neq j$ does not contain the variable x_i. Thus, for each i exactly one of the terms $t_{x_i}^+$ and $t_{x_i}^-$ is contained in E and the other is contained in S. We define the assignment α where $\alpha(x_i) = \top$ iff $t_{x_i}^+ \in S$. It remains to show that α is a satisfying assignment for ϕ. So let C_j be some clause of ϕ. Since $y_j \in S'$ we know that there is some $t \in S$ with $t \succ^{gmpo} y_j$, i.e., $y_j \in \mathcal{V}(t)$. There are two cases. First, if $t = t^+_{x_i}$ for some i, then by the definition of t^+ we know that $y_j \in \mathcal{V}(t^+_{x_i})$ implies $x_i \in C_j$, and hence C_j is evaluated to true, since $\alpha(x_i) = \top$ by definition of α. Otherwise, $t = t^-_{x_i}$ for some i where now $\overline{x_i} \in C_j$. Moreover, as $t^-_{x_i} \in S$, we know that $t^+_{x_i} \notin S$ and hence, $\alpha(x_i) = \bot$. Together with $\overline{x_i} \in C_j$ this again shows that C_j is evaluated to true. Hence, all clauses evaluate to true using α which proves that ϕ is satisfiable. ◀

The following corollary states that NP-completeness is essentially due to fact that $\succ_{gms}$ and $\succsim_{gms}$ are hard to compute, even if all comparisons of the elements in the multiset are given. It can be seen within the previous proof, where the important part of the reduction from SAT was to define for each formula ϕ the multisets L and R such that ϕ is satisfiable iff $L \succ^{gmpo}_{gms} R$ (and also iff $L \succsim^{gmpo}_{gms} R$).

▶ **Corollary 4.3.** *Given two orderings $\succ$ and $\succsim$, two multisets M and N, and the set $\{(x, y, x \succ y, x \succsim y) \mid x \in M, y \in N\}$, deciding $M \succ_{gms} N$ and $M \succsim_{gms} N$ is NP-complete.*

Of course, if the splitting for the multiset comparison is given, then deciding $\succ_{gms}$ and $\succsim_{gms}$ becomes polynomial.

All our results have also been generalized to RPO where for every function symbol there is a status function τ which determines whether the arguments of each function f should be compared lexicographically ($\tau(f) = \mathit{lex}$) or via multisets ($\tau(f) = \mathit{mul}$).[3] Here, the existing inference rules of $\succ^{gmpo}$ are modified that instead of $f \approx g$ it is additionally demanded that $\tau(f) = \tau(g) = \mathit{mul}$. Moreover, there are two additional inference rules for $f \approx g$ and $\tau(f) = \tau(g) = \mathit{lex}$, one for the strict ordering $\succ^{grpo}$ and one for the non-strict ordering $\succsim^{grpo}$.

Again, $\succ^{grpo}$ is used in AProVE instead of the standard definition of RPO ($\succ^{rpo}$,[9]). However, during our formalization we have detected that in contrast to $(\succsim^{gmpo}, \succ^{gmpo})$, the pair $(\succsim^{grpo}, \succ^{grpo})$ is not a reduction pair, as the orderings are not stable. To see this, consider a precedence where a and b have least precedence, and $\tau(\mathsf{a}) \neq \tau(\mathsf{b})$. Then $x \succsim^{grpo} \mathsf{a}$, but $\mathsf{b} \not\succsim^{grpo} \mathsf{a}$.

Our solution to this problem is to add the following further inference rules which allow comparisons of terms $f(\vec{s})$ with $g(\vec{t})$ where $\tau(f) \neq \tau(g)$. In detail, we require that $\vec{t}$ is empty and for a strict decrease, additionally $\vec{s}$ must be non-empty.

$$\frac{|\vec{s}| > 0 \quad |\vec{t}| = 0}{f(\vec{s}) \succ^{grpo} g(\vec{t})}\, f \approx g, \tau(f) \neq \tau(g) \qquad \frac{|\vec{t}| = 0}{f(\vec{s}) \succsim^{grpo} g(\vec{t})}\, f \approx g, \tau(f) \neq \tau(g)$$

If these inference rules are added, then indeed $(\succsim^{grpo}, \succ^{grpo})$ is a monotone reduction pair. As a consequence, one can interpret AProVE's version of RPO as a sound, but non-stable under-approximation of $\succ^{grpo}$.

We also tried to to relax the preconditions further, e.g. by allowing vectors $\vec{t}$ of length at most one. But no matter whether we also restrict the length of $\vec{s}$ in some way or not, and no matter whether we compared the arguments $\vec{s}$ and $\vec{t}$ lexicographically or via multisets, the outcome was always that transitivity or strong normalization are lost.

For example, if we add the inference rule that $\{\{\vec{s}\}\} \succ^{grpo}_{gms} \{\{\vec{t}\}\}$, $|\vec{t}| \leq 1$, $f \approx g$, and $\tau(f) \neq \tau(g)$ implies $f(s) \succ^{grpo} g(t)$, then strong normalization is lost: assume the precedence

[3] We do not consider permutations for lexicographic comparisons in the definition of RPO as this feature can be simulated by generating reduction pairs using an RPO (without permutations) in combination with argument filterings as defined in [1]. In this way, we only have to formalize permutations once and we can reuse them for other orderings like KBO.

is defined by $\mathsf{f} \approx \mathsf{g} \approx \mathsf{h}$ and $3 > 2 > 1 > 0$, and the status is defined by $\tau(\mathsf{f}) = \tau(\mathsf{h}) = \mathit{lex}$ and $\tau(\mathsf{g}) = \mathit{mul}$. Then $\mathsf{f}(0,3) \succ^{grpo} \mathsf{g}(2) \succ^{grpo} \mathsf{h}(1) \succ^{grpo} \mathsf{f}(0,3)$ clearly shows that the resulting ordering is not strongly normalizing anymore.

To summarize, $\succ^{gmpo}$ and $\succ^{grpo}$ are strictly more powerful reduction orderings than the standard definitions of MPO and RPO ($\succ^{mpo}$ and $\succ^{rpo}$). The price for the increased power is that checking constraints for $\succ^{gmpo}$ and $\succ^{grpo}$ becomes NP-complete whereas it is in P for $\succ^{mpo}$ and $\succ^{rpo}$.

Note that if one would provide all required splittings for the multiset comparisons in $\succ^{gmpo}$ and $\succ^{grpo}$, then constraint checking again becomes polynomial. However, this would make certificates more bulky, and since in practice the arities of function symbols are rather small, certification can efficiently be done even without additional splitting information.

5 SCNP Reduction Pairs

The size-change criterion of [19] to prove termination of programs can be seen as a graph-theoretical problem: given a set of graphs—encoding for each recursive call the decrease in size of each argument—one tries to decide the *size-change termination condition* (SCT condition) on the graphs, namely that in every infinite sequence of graphs one can find an argument whose size is strictly decreased infinitely often. If the condition is satisfied, then termination is proved.

Concerning automation of the size-change principle, there are two problems: first, the base ordering (or size-measure or ranking function) must be provided to construct the graphs, and second, even for a given base ordering, deciding the SCT condition is PSPACE-complete. To overcome these problems, in [3] and [6] sufficient criteria have been developed. They approximate the SCT condition in a way that can be encoded into SAT.

Another benefit of [6] is an integration of the approximated SCT condition as a reduction pair in the dependency pair framework (DP framework) [12], called *SCNP reduction pair*.

Although IsaFoR contains already a full formalization of size-change termination as it is used in [23], an integration of SCNP reduction pairs in the certification process might be beneficial for two reasons:

- since deciding the SCT condition is PSPACE hard, whereas the approximated condition can be encoded into SAT, the certificates might be easier to check
- the approximated SCT condition is not fully subsumed by the SCT condition as only the former allows incremental termination proofs in the DP framework

Before we describe our formalization of SCNP reduction pairs, we shortly recall some notions of the DP framework, a popular framework to perform modular termination proofs for TRSs. The main data structure are *DP problems* $(\mathcal{P}, \mathcal{R})$ consisting of two TRSs where all rules in $\mathcal{P}$ are of the form $F(\dots) \to G(\dots)$ where F, G are symbols that do not occur in $\mathcal{R}$. The main task is to prove finiteness of the a given DP problem $(\mathcal{P}, \mathcal{R})$, i.e., absence of *infinite minimal chains* $s_1\sigma \to t_1\sigma \to_{\mathcal{R}}^* s_2\sigma \to t_2\sigma \dots$ where all $s_i \to t_i \in \mathcal{P}$ and all $t_i\sigma$ are terminating w.r.t. $\mathcal{R}$. To this end, one uses various *processors* to simplify the initial DP problem for a TRS until the $\mathcal{P}$-component is empty.

One of the most important processors is the reduction pair processor [12, 14]—where here we only present its basic version without other refinements like usable rules w.r.t. an argument filtering [12]. It can remove strictly decreasing rules from $\mathcal{P}$, provided that both $\mathcal{P}$ and $\mathcal{R}$ are at least weakly decreasing.

▶ **Theorem 5.1** (Reduction pair processor). *Let $(\succsim, \succ)$ be a reduction pair. If $\mathcal{P} \subseteq \succ \cup \succsim$ and $\mathcal{R} \subseteq \succsim$ then $(\mathcal{P}, \mathcal{R})$ is finite if $(\mathcal{P} \setminus \succ, \mathcal{R})$ is finite.*

Hence, to prove termination it suffices to find different reduction pairs to iteratively remove rules from $\mathcal{P}$ until all rules of $\mathcal{P}$ have been removed. Thus, with SCNP reduction pairs it is possible to choose different base orderings to remove different rules of $\mathcal{P}$.

In the following we report on details of SCNP reduction pairs as defined in [6] and on their formalization. Essentially, SCNP reduction pairs are generated from reduction pairs $(\succsim, \succ)$ via a multiset extension of a lexicographic combination of $\succ$ with the standard ordering on the naturals.

▶ **Definition 5.2** (Multiset extension). We define that μ is a *multiset extension* iff for each ordering pair $(\succsim, \succ)$ over A, the pair $(\succsim_\mu, \succ_\mu)$ is an ordering pair over $\mathfrak{P}(A)$. Moreover, whenever $\succ$ and $\succsim$ are closed under an operator *op* ($x \underset{(\sim)}{\succ} y$ implies $op(x) \underset{(\sim)}{\succ} op(y)$ for all x, y), then $\succ_\mu$ and $\succsim_\mu$ must also be closed under the image of *op* on multisets ($M \underset{(\sim)}{\succ}_\mu N$ implies $op[M] \underset{(\sim)}{\succ}_\mu op[N]$).

We also call $(\succsim_\mu, \succ_\mu)$ the *multiset extension* of $(\succsim, \succ)$ w.r.t. μ.

For example, *gms* is a multiset extension and in [3] and [6] in total four extensions are listed to compare multisets (*gms*, *min*, *max*, and *dms*). The extension *dms* is the dual multiset extension [4] where our formulation is equivalent to the definition in [6].[4] The strict relation is defined as $M \succ_{dms} N$ iff $M = \{\{x_1, \ldots, x_m\}\} \cup \{\{z_1, \ldots, z_k\}\}$, $N = \{\{y_1, \ldots, y_n\}\} \cup \{\{z'_1, \ldots, z'_k\}\}$, $n > 0$, $\forall i. z_i \succsim z'_i$, and $\forall x_i. \exists y_j. x_i \succ y_j$; the non-strict relation $\succsim_{dms}$ is defined like $\succ_{dms}$ except that the condition $n > 0$ is omitted.

For SCNP reduction pairs, multisets are compared via one of the four multiset extensions. And to generate multisets from terms, the notion of a level mapping is used.

▶ **Definition 5.3** (Level mapping [6]). For each $f \in \mathcal{F}$ with arity n, let $\pi(f) \in \mathfrak{P}(\{1, \ldots, n\} \times \mathbb{N})$.[5] We define the *level-mapping* $\mathcal{L} : \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathfrak{P}(\mathcal{T}(\mathcal{F}, \mathcal{V}) \times \mathbb{N})$ where $\mathcal{L}(f(s_1, \ldots, s_n)) = \{\{\langle s_i, k\rangle \mid (i, k) \in \pi(f)\}\}$.[6]

▶ **Definition 5.4.** Let $(\succsim, \succ)$ be a reduction pair for terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$. Let μ be a multiset extension. The ordering pair $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ over $\mathcal{T}(\mathcal{F}, \mathcal{V}) \times \mathbb{N}$ is defined as the lexicographic combination of $(\succsim, \succ)$ with the $>$-ordering on the naturals: $\langle s, n\rangle \succsim^{\mathbb{N}} \langle t, m\rangle$ iff $s \succ t \vee (s \succsim t \wedge n \geq m)$, and $\langle s, n\rangle \succ^{\mathbb{N}} \langle t, m\rangle$ iff $s \succ t \vee (s \succsim t \wedge n > m)$. The ordering pair $(\succsim^{\mathbb{N}}_\mu, \succ^{\mathbb{N}}_\mu)$ is defined as the multiset extension of $(\succsim^{\mathbb{N}}, \succ^{\mathbb{N}})$ w.r.t. μ.

In principle, $\succ^{\mathbb{N}}_\mu \cup \succsim^{\mathbb{N}}_\mu$ is the part of a SCNP reduction pair that should be used to compare left- and right-hand sides of $\mathcal{P}$ within the reduction pair processor. However, one also needs to orient the rules in $\mathcal{R}$ via $\succsim$. To this end, two types are introduced in [6] so that the final ordering incorporates both $\succ^{\mathbb{N}}_\mu \cup \succsim^{\mathbb{N}}_\mu$ for $\mathcal{P}$ and $\succsim$ for $\mathcal{R}$. In detail, the signature $\mathcal{F}$ is partitioned into $\mathcal{F}^b \uplus \mathcal{F}^\sharp$ consisting of base symbols $\mathcal{F}^b$ and tuple-symbols $\mathcal{F}^\sharp$. The set of *base terms* is $\mathcal{T}(\mathcal{F}^b, \mathcal{V})$, and a *tuple term* is a term of the form $F(t_1, \ldots, t_n)$ where $F \in \mathcal{F}^\sharp$ and each t_i is a base term.

Notice that for a DP problem $(\mathcal{P}, \mathcal{R})$ all terms in $\mathcal{P}$ are tuple terms and all terms in $\mathcal{R}$ are base terms if one chooses $\mathcal{F}^\sharp$ to be the set of root symbols of terms in $\mathcal{P}$. Therefore, SCNP reduction pairs are defined in a way that the ordering depends on whether tuple terms or base terms are compared.

[4] There is a difference in the definition of the dual multiset extension in [3,4] to the definition in [6] which is similar to the difference between $\succ_{ms}$ and $\succ_{gms}$.

[5] In [6] there was an additional condition that $\pi(f)$ may not contain two entries $\langle j, k_1\rangle$ and $\langle j, k_2\rangle$. It turned out that this condition is not required for soundness.

[6] We use the notation $\mathcal{L}$ instead of ℓ for level mappings as in this paper, ℓ are left-hand sides of rules.

▶ **Definition 5.5** (SCNP reduction pair [6])**.** Let μ be a multiset extension and $\mathcal{L}$ be a level mapping. Let $(\succsim, \succ)$ be a reduction pair over $\mathcal{T}(\mathcal{F}, \mathcal{V})$. The *SCNP reduction pair* is the pair $(\succsim^{\mathcal{L},\mu}, \succ^{\mathcal{L},\mu})$ where the relations $\succsim^{\mathcal{L},\mu}$ and $\succ^{\mathcal{L},\mu}$ over $\mathcal{T}(\mathcal{F}, \mathcal{V})$ are defined as follows. If t and s are tuple terms, then $t \succ^{\mathcal{L},\mu} s$ iff $\mathcal{L}(t) \succ_{\mu}^{\mathbb{N}} \mathcal{L}(s)$, and $t \succsim^{\mathcal{L},\mu} s$ iff $\mathcal{L}(t) \succsim_{\mu}^{\mathbb{N}} \mathcal{L}(s)$. Otherwise, if t and s are base terms, then $t \succsim^{\mathcal{L},\mu} s$ iff $t \succsim s$, and $t \succ^{\mathcal{L},\mu} s$ iff $t \succ s$.

Before stating the major theorem of [6] that SCNP reduction pairs are reduction pairs, we first have to clarify the notion of reduction pair: In [6, Sec. 2] a non standard definition of a reduction pair is used which differs from Def. 2.1. To distinguish between both kinds of reduction pairs we call the ones of [6] *typed reduction pairs.*

▶ **Definition 5.6** (Typed reduction pair [6])**.** A *typed reduction pair* is a reduction pair $(\succsim, \succ)$ over $\mathcal{T}(\mathcal{F}, \mathcal{V})$ with the additional condition that $\succsim$ compares either two tuple terms or two base terms.

So, the major theorem of [6] states that whenever $(\succsim, \succ)$ is a typed reduction pair, then so is $(\succsim^{\mathcal{L},\mu}, \succ^{\mathcal{L},\mu})$ where μ is one of the four mentioned multiset extensions.

The major problem in the formalization of exactly this theorem required a link between the two notions of reduction pairs since all other theorems working with reduction pairs in IsaFoR are using Def. 2.1.

At this point, it turned out that there are problems with Def. 5.6. Of course, to use the major theorem of [6] one needs a typed reduction pair to start from. Unfortunately, common reduction pairs like RPO are not typed reduction pairs w.r.t. Def. 5.6. For example, if $F \in \mathcal{F}^\sharp$ then $F(F(x)) \succsim^{grpo} F(F(x))$, but then $\succsim$ does not satisfy the additional condition of Def. 5.6, since here two terms are in relation which are neither base terms nor tuple terms.

A possible solution might be to require that $(\succsim, \succ)$ is just a reduction pair and then show that $(\succsim^{\mathcal{L},\mu}, \succ^{\mathcal{L},\mu})$ is a typed reduction pair. However, even with this adaptation the problems remain, since Def. 5.6 itself is flawed: assume $(\succsim, \succ)$ is a typed reduction pair and F is a non-constant tuple symbol. Since $\succsim$ is a quasi-ordering (on tuple-terms) it is reflexive, and thus $F(\vec{t}) \succsim F(\vec{t})$ for every list $\vec{t}$ of base terms. By monotonicity of $\succsim$ also $F(\dots, F(\vec{t}), \dots) \succsim F(\dots, F(\vec{t}), \dots)$ must hold, in contradiction to the condition that $\succsim$ only compares base terms or tuple terms. So, there is a severe problem in demanding both monotonicity of $\succsim$ and the additional condition of Def. 5.6.

Repairing Def. 5.6 by only requiring monotonicity w.r.t. $\mathcal{F}^b$ is also no solution, since for using reduction pairs in the reduction pair processor, it is essential that $\succsim$ is also closed under $\mathcal{F}^\sharp$-contexts.

For a proper fix of SCNP reduction pairs, note that the distinction between tuple terms and base terms in [6] is solely performed, to have two kinds of orderings: $\succsim$ for orienting rules in $\mathcal{R}$, and $\succsim_{\mu}^{\mathbb{N}}$ and $\succ_{\mu}^{\mathbb{N}}$ for orienting rules of $\mathcal{P}$.

However, there is already a notion which allows the usage of different orderings for orientation of $\mathcal{P}$ and $\mathcal{R}$, namely *reduction triples.* The advantage of this notion is the fact that it does not require any distinction between base and tuple terms.

▶ **Definition 5.7** (Reduction triple, [14])**.** A *reduction triple* is a triple $(\succsim, \succsim_\top, \succ_\top)$ such that $(\succsim, \succ_\top)$ is a reduction pair and $(\succsim_\top, \succ_\top)$ is a non-monotone reduction pair.

Reduction triples can be used instead of reduction pairs in Thm. 5.1: in [14] it is shown that whenever $\mathcal{P} \subseteq \succ_\top \cup \succsim_\top$ and $\mathcal{R} \subseteq \succsim$ then $\mathcal{P}$ can be replaced by $\mathcal{P} \setminus \succ_\top$ in the DP problem $(\mathcal{P}, \mathcal{R})$. The proof is similar to the one of Thm. 5.1 and also in IsaFoR it was easy to switch from reduction pairs to reduction triples. Hence, the obvious attempt is to define SCNP reduction pairs as reduction triples. As there is no distinction of base terms and tuple terms, we will not run into problems that are caused by the required monotonicity of $\succsim$.

▶ **Definition 5.8** (SCNP reduction triple)**.** Let μ be a multiset extension and $\mathcal{L}$ be a level mapping. Let $(\succsim, \succ)$ be a reduction pair.

We define $\succsim_\top$ and $\succ_\top$ as $t \succsim_\top s$ iff $\mathcal{L}(t) \succsim_\mu^{\mathbb{N}} \mathcal{L}(s)$, and $t \succ_\top s$ iff $\mathcal{L}(t) \succ_\mu^{\mathbb{N}} \mathcal{L}(s)$. Then the *SCNP reduction triple* is defined as $(\succsim, \succsim_\top, \succ_\top)$.

If we are able to prove that every SCNP reduction triple is indeed a reduction triple, then we are done. Unfortunately, it turns out that an SCNP reduction triple is not a reduction triple, where the new problem is that compatibility between $\succ_\top$ and $\succsim$ cannot be ensured. As an example, consider a reduction pair $(\succsim, \succ)$ which is defined via an RPO with precedence $b > a$ and status $\tau(F) = \mathit{lex}$. Moreover, let the level-mapping be defined via $\pi(F) = \{\{\langle 2, 0\rangle\}\}$ and take $\mu = \mathit{gms}$. Then $F(a,b) \succ_\top F(b,a)$ since $\mathcal{L}(F(a,b)) = \{\{\langle b, 0\rangle\}\} \succ_{\mathit{gms}}^{\mathbb{N}} \{\{\langle a, 0\rangle\}\} = \mathcal{L}(F(b,a))$. Furthermore, $F(b,a) \succsim F(a,b)$. However, if $\succ_\top$ and $\succsim$ were compatible, then we would be able to conclude $F(a,b) \succ_\top F(a,b)$, a contradiction.

To this end, we finally have defined a weaker notion than reduction triples which can still be used like reduction triples.

▶ **Definition 5.9** (Root reduction triple)**.** A *root reduction triple* is a triple $(\succsim, \succsim_\top, \succ_\top)$ such that $(\succsim_\top, \succ_\top)$ is a non-monotone reduction pair, $\succsim$ is a stable and monotone quasi-ordering, and whenever $s \succsim t$ then $f(v_1, \ldots, v_{i-1}, s, v_{i+1}, \ldots, v_n) \succsim_\top f(v_1, \ldots, v_{i-1}, t, v_{i+1}, \ldots, v_n)$.

Note that every reduction triple $(\succsim, \succsim_\top, \succ_\top)$ is also a root reduction triple provided that $\succsim \subseteq \succsim_\top$—and as far as we know this condition is satisfied for all reduction triples that are currently used in termination tools. Moreover, root reduction triples can be used in the same way as reduction triples to remove pairs which has been proved in IsaFoR.

▶ **Theorem 5.10** (Root reduction triple processor)**.** *Let $(\succsim, \succsim_\top, \succ_\top)$ be a root reduction triple. Whenever $\mathcal{P} \subseteq \succ_\top \cup \succsim_\top$, $\mathcal{R} \subseteq \succsim$, and $(\mathcal{P} \setminus \succ_\top, \mathcal{R})$ is finite, then $(\mathcal{P}, \mathcal{R})$ is finite.*

Hence, the notion of root reduction triple seems useful for termination proving. And indeed, it turns out that each SCNP reduction triple is a root reduction triple which finally shows that SCNP reduction triples can be used to remove pairs from DP problems.[7]

▶ **Theorem 5.11.** *Every SCNP reduction triple is a root reduction triple.*

Thm. 5.11 is formally proved within IsaFoR, theory *SCNP*. Note that this formalization was straightforward, once the notion of root reduction triple was available: the whole formalization takes only 310 lines. It also includes the feature of ϵ-arguments—an extension of size-change graphs which is mentioned in both [23] and [6]—and it contains results on C_e-compatibility and compatibility w.r.t. to argument filterings. The latter results are important when dealing with usable rules, cf. [12] for further details.

Regarding the formalization of multiset extensions—which is orthogonal to the formalization of SCNP reduction triple—we were able to integrate three of the four mentioned multiset extensions. However, for the multiset extension *dms* it is essential that the signature $\mathcal{F}$ is finite as otherwise strong normalization is not necessarily preserved, cf. Ex. 5.12. Here, the essential issue is that without an explicit bound on the sizes of the multiset, strong normalization is lost (such a bound is explicitly demanded in [3], but not in [6]).

▶ **Example 5.12.** In [6] it is assumed that the *initial* TRS is finite. Hence, also the initial signature is finite which in turn gives a bound on the sizes of the constructed multisets.

[7] An alternative—but unpublished—fix to properly define SCNP reduction pairs has been developed by Carsten Fuhs. It is based on typed term rewriting. (private communication)

However, for the soundness of SCNP reduction pairs it is essential that the signature of the *current* system is finite, since otherwise the following steps would be a valid termination proof for the TRS $\mathcal{R} = \{\mathsf{a} \to \mathsf{a}\}$ as there is no bound on the sizes of multisets.

- Build the initial DP problem $(\{\mathsf{A} \to \mathsf{A}\}, \mathcal{R})$.
- Replace this DP problem by the DP problem $\mathcal{D} = (\{\mathsf{A}_i(x, \ldots, x) \to \mathsf{A}_{i+1}(x, \ldots, x) \mid i \in \mathbb{N}\}, \emptyset)$ where the arity of each A_i is i. This step is sound, as $\mathcal{D}$ is not finite.
- Replace $\mathcal{D}$ by $(\emptyset, \emptyset)$. This can be done using the SCNP reduction pair where $\mu = \mathit{dms}$, $\pi(\mathsf{A}_i) = \{\langle j, 0\rangle \mid 1 \le j \le i\}$, and $(\succsim, \succ)$ is any reduction pair. The reason is that

$$\{\{\underbrace{\langle x,0\rangle, \ldots, \langle x,0\rangle}_{i \text{ times}}\}\} \succ^{\mathit{dms}} \{\{\underbrace{\langle x,0\rangle, \ldots, \langle x,0\rangle}_{i+1 \text{ times}}\}\}.$$

The demand for a bound on the signature for *dms* resulted in a small problem in our formalization, since in IsaFoR we never have finite signatures: for each symbol f we directly include infinitely many f's, one for each possible arity in $\mathbb{N}$. So, in principle there might not be any bound on the size of the multisets that are constructed by the level mapping. Hence, we use a slightly different version of *dms* which includes some fixed bound n on the size of the multisets: we define $M \underset{(\sim)}{\succ}_{\mathit{dms}\text{-}n} N$ iff $M \underset{(\sim)}{\succ}_{\mathit{dms}} N$ and $|N| \le n \vee |N| = |M|$. Hence, *dms-n* is a restriction of *dms* where either the sizes of the multisets are bounded by n or where multisets of the same sizes are compared.

Note that the additional explicit restriction of $|N| \le n$ ensures strong normalization even for infinite $\mathcal{F}$ or unbounded multisets. The other alternative $|N| = |M|$ is added, as otherwise reflexivity of $\succsim_{\mathit{dms}\text{-}n}$ is lost, for example $\{\{\underbrace{x, \ldots, x}_{n+1}\}\} \succsim_{\mathit{dms}\text{-}n} \{\{\underbrace{x, \ldots, x}_{n+1}\}\}$ would no longer hold.

In practice, the difference between *dms* and *dms-n* can be neglected. As for the certification we only consider finite TRSs, we just precompute a suitable large enough number n such that for the resulting constraints *dms* and *dms-n* coincide.

We conclude this section by showing that certification of SCNP reduction triples is NP-hard in the case of $\mu \in \{\mathit{gms}, \mathit{dms}\}$ which also implies that deciding $\succ_{\mathit{dms}}$ is NP-hard.

▶ **Theorem 5.13.** *Given a SCNP reduction triple $(\succsim, \succsim_\top, \succ_\top)$ and $\mu \in \{\mathit{gms}, \mathit{dms}\}$, the problem of deciding $\ell \succ_\top r$ for two terms ℓ and r is NP-hard.*

Proof. For $\mu = \mathit{gms}$ we use nearly the identical reduction from SAT as we used in the proof of Thm. 4.2. To be more precise, for each formula ϕ we use the exactly the same terms ℓ and r and the same multisets L and R as before. Moreover, we define the level-mapping by choosing $\pi(\mathsf{g}) = \{\langle i, 0\rangle \mid 1 \le i \le 2n\}$ and $\pi(\mathsf{h}) = \{\langle i, 0\rangle \mid 1 \le i \le n+m\}$. Then for $L' = \{\{\langle s, 0\rangle \mid s \in L\}\}$ and $R' = \{\{\langle s, 0\rangle \mid s \in R\}\}$ we obtain $\ell \succ_\top r$ iff $L' \succ^{\mathbb{N}}_{\mathit{gms}} R'$ iff $L \succ_{\mathit{gms}} R$. It remains to choose $\succ$ as the MPO within the proof of Thm. 4.2. Then $L \succ_{\mathit{gms}} R$ iff ϕ is satisfiable, so in total, $\ell \succ_\top r$ iff ϕ is satisfiable.

Furthermore, it can easily be argued that NP-hardness is not a result of our extended definition of MPO: we also achieve $L \succ_{\mathit{gms}} R$ iff ϕ is satisfiable if we define $\succ$ by a polynomial interpretation $\mathcal{P}ol$ with $\mathcal{P}ol(\mathsf{f}(x_0, \ldots, x_m)) = 1 + x_0 + \cdots + x_m$ and $\mathcal{P}ol(\mathsf{a}) = 0$.

For $\mu = \mathit{dms}$ one can use a similar reduction from SAT: satisfiability of ϕ is equivalent to $\ell \succ_\top r$ using the same level-mapping as before, but where now $\ell = \mathsf{h}(x_1 \vee \overline{x_1}, \ldots, x_n \vee \overline{x_n}, \mathsf{s}(C'_1), \ldots, \mathsf{s}(C'_m))$, $r = \mathsf{g}(x_1, \overline{x_1}, \ldots, x_n, \overline{x_n})$, and $\succ$ is defined as the polynomial interpretation $\mathcal{P}ol$ where $\mathcal{P}ol(\mathsf{s}(x)) = 1 + x$ and $\mathcal{P}ol(x \vee y) = x + y$. Here, each C'_i is a representation of clause C_i as disjunction. ◀

6 Certification Algorithms

For the certification of existing proofs using RPO and SCNP reduction pairs there are in principle two possibilities, which we will explain using RPO.

The first solution for certifying $s \succ^{grpo} t$ is to use a shallow embedding. In this approach, some untrusted tool figures out how the inference rules of RPO have to be applied and generates a proof script that can then be checked by the proof assistant. This solution cannot be used in our case, since CeTA is obtained from IsaFoR via code generation.

The second solution is to use a deep embedding where an algorithm for deciding RPO is developed within the proof assistant in combination with a soundness proof. Then this algorithm is amenable for code generation, and such an algorithm is also used in related certifiers [5, 7] where it is accessible via reflection. Hence, we had to develop a function for deciding RPO constraints within Isabelle. In our case, we have written a function $grpo^{\tau}_{>}$ for RPO, which is parametrized by a precedence $>$ and a status τ. It takes two terms s and t as input and returns the pair $(s \succ^{grpo} t, s \succsim^{grpo} t)$. In fact, we even defined RPO via $grpo^{\tau}_{>}$ and only later on derived the inference rules that have been presented in Sec. 4.

Since we proved several properties of RPO directly via $grpo^{\tau}_{>}$ (theory *RPO*), we implemented $grpo^{\tau}_{>}$ in a straightforward way as recursive function. As a result, $grpo^{\tau}_{>}$ has exponential runtime, since no sharing of identical subcalls is performed. To this end, we developed a second function for RPO, that has been proved to be equivalent to $grpo^{\tau}_{>}$, but where memoization is integrated—a well known technique where intermediate results are stored to avoid duplicate computations. In principle, this is an easy programming task, but since we use a deep embedding, we had to formally prove correctness of this optimization.

To stay as general as possible, the memoized function was implemented independent of the actual data structure used as memory. The interface we use for a memory is as follows. We call a memory valid w.r.t. to a function f if all entries in the memory are results of f. Moreover we require functionality for looking up a result in the memory and for storing a new result with the obvious soundness properties: storing a correct entry in a valid memory yields a valid memory and looking up an entry in a valid memory returns a correct result, i.e., the same result that would have been computed by f.

Assuming we have such a memory at our disposal the idea of memoizing is straightforward: before computing a result we do a lookup in the memory and if an entry is found we return it and leave the memory unchanged. Otherwise we compute as usual, store the result in the memory and return both.

The main difficulty was that all recursive calls of $grpo^{\tau}_{>}$ are indirect via higher-order functions like the computation of the multiset- and lexicographic extension of a relation. Consequently results of recursive calls to RPO are not available directly, but only in these higher-order functions. Thus, they have to take care of storing results in the memory and of passing it on to the next RPO call. Hence, new memoizing versions of all these higher-order functions had to be implemented and their soundness had to be proved. Soundness meaning that, when given equivalent functions as arguments, they compute the same results as their counterparts without memory handling, and that given a valid memory as input they return a valid memory in addition to their result. For further details we refer to theory *Efficient-RPO*.

Concerning the required decision procedures for *gms* and *dms* which have to find suitable splittings, we used a branch-and-bound approach.

We tested our certification algorithms by rerunning all experiments that have been performed in [6]. Here, AProVE tries to prove termination of 1,381 TRSs from the termination problem database (TPDB version 7.0) using 20 different strategies where in 12 cases SCNP reduction pairs are used and full size-change termination is tried 4 times. Here, in 16 strategies RPO or weaker orderings have been tried. As in [6] we used a timeout of 60 seconds and although we used a different computer than in [6], on the $20 \times 1,381$ termination problems, there are only 8 differences in both experiments—all due to a timeout.

By performing the experiments we were able to detect a bug in the proof output of AProVE—the usable rules have not been computed correctly. After a corresponding fix indeed all 9,025 generated proofs could be certified by CeTA (version 2.2).

The experiments contain 432 proofs where the usage of $\succ_{gms}$ and $\succsim_{gms}$ was essential, i.e., where the constraints could not be oriented by $\succ_{ms}$ and $\succeq_{ms}$. If one allows equality modulo the equivalence relation induced by the ordering, then still 39 proofs require $\succ_{gms}$ and $\succsim_{gms}$.

Regarding the time required for certification, although it is NP-complete, in our experiments, certification is much faster than proof search. The reason is that our certification algorithms are only exponential in the arity of the function symbols, and in the experiments the maximal arity was 7. In numbers: AProVE required more than 31 hours (≈ 4 seconds per example) whereas CeTA was done in below 6 minutes (≈ 0.04 seconds per example).

The experiments also show that proofs using SCNP reduction pairs can indeed be certified faster than proofs using full size-change termination. In average, the latter proofs require 50 % more time for certification than the former.

All details of our experiments are available at `http://cl-informatik.uibk.ac.at/software/ceta/experiments/multisets/`.

7 Summary

We have studied the generalization of the multiset ordering which is generated by two orderings: a strict one and a compatible non-strict one. Indeed this generalization preserves most properties of the standard multiset ordering, where we only detected one difference: the decision problem becomes NP-complete.

Concerning termination techniques that depend on multiset orderings, we formalized and corrected an extended variant of RPO that is used within AProVE, and we formalized and corrected SCNP reduction pairs. Certification of both techniques is NP-hard.

Acknowledgments

We thank Carsten Fuhs for helpful discussions and his support in the experiments, and we thank the anonymous referees for their helpful comments.

References

1 T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theor. Comput. Sci.*, 236(1-2):133–178, 2000.

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

3 A. M. Ben-Amram and M. Codish. A SAT-based approach to size change termination with global ranking functions. In *Proc. TACAS '08*, volume 4963 of *LNCS*, pages 218–232, 2008.

4 A. M. Ben-Amram and C. Soon Lee. Program termination analysis in polynomial time. *ACM Trans. Program. Lang. Syst.*, 29(1), 2007.

5 F. Blanqui and A. Koprowski. CoLoR: a Coq library on well-founded rewrite relations and its application on the automated verification of termination certificates. *Mathematical Structures in Computer Science*, 21(4):827–859, 2011.

6 M. Codish, C. Fuhs, J. Giesl, and P. Schneider-Kamp. Lazy abstraction for size-change termination. In *Proc. LPAR '10*, volume 6397 of *LNCS*, pages 217–232, 2010.

7 E. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Certification of automated termination proofs. In *Proc. FroCoS '07*, volume 4720 of *LNAI*, pages 148–162, 2007.

8 N. Dershowitz. Orderings for term-rewriting systems. *Theor. Comput. Sci.*, 17:279–301, 1982.

9 N. Dershowitz. Termination of rewriting. *J. Symb. Comp.*, 3(1-2):69–116, 1987.

10 N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Comm. ACM*, 22(8):465–476, 1979.

11 J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. IJCAR '06*, volume 4130 of *LNAI*, pages 281–286, 2006.

12 J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *J. Autom. Reason.*, 37(3):155–203, 2006.

13 F. Haftmann and T. Nipkow. Code generation via higher-order rewrite systems. In *Proc. FLOPS '10*, volume 6009 of *LNCS*. Springer, 2010.

14 N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: Techniques and features. *Inf. Comput.*, 205(4):474–511, 2007.

15 J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proc. LICS '99*, pages 402–411. IEEE Computer Society Press, 1999.

16 A. Koprowski. Coq formalization of the higher-order recursive path ordering. *Appl. Algebra Eng. Commun. Comput.*, 20(5-6):379–425, 2009.

17 K. Korovin and A. Voronkov. Orienting rewrite rules with the Knuth-Bendix order. *Inf. Comput.*, 183(2):165–186, 2003.

18 M. S. Krishnamoorthy and P. Narendran. On recursive path ordering. *Theor. Comput. Sci.*, 40:323–328, 1985.

19 C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *Proc. POPL '01*, pages 81–92, 2001.

20 T. Nipkow. An inductive proof of the wellfoundedness of the multiset order, 1998. Available at `http://www4.in.tum.de/~nipkow/misc/multiset.ps`.

21 T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, Berlin-Heidelberg, 2002.

22 P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving termination using recursive path orders and SAT solving. In *Proc. FroCoS '07*, volume 4720 of *LNAI*, pages 267–282, 2007.

23 R. Thiemann and J. Giesl. The size-change principle and dependency pairs for termination of term rewriting. *Appl. Alg. Eng. Comm. Comput.*, 16(4):229–270, 2005.

24 R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proc. TPHOLs '09*, volume 5674 of *LNCS*, pages 452–468, 2009.