Ralf Treinen (Ed.)

LNCS 5595

Rewriting Techniques and Applications

20th International Conference, RTA 2009
Brasília, Brazil, June/July 2009
Proceedings

Lecture Notes in Computer Science 5595

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison
Lancaster University, UK

Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler
University of Surrey, Guildford, UK

Jon M. Kleinberg
Cornell University, Ithaca, NY, USA

Alfred Kobsa
University of California, Irvine, CA, USA

Friedemann Mattern
ETH Zurich, Switzerland

John C. Mitchell
Stanford University, CA, USA

Moni Naor
Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz
University of Bern, Switzerland

C. Pandu Rangan
Indian Institute of Technology, Madras, India

Bernhard Steffen
University of Dortmund, Germany

Madhu Sudan
Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos
University of California, Los Angeles, CA, USA

Doug Tygar
University of California, Berkeley, CA, USA

Gerhard Weikum
Max-Planck Institute of Computer Science, Saarbruecken, Germany

This page intentionally left blank

Ralf Treinen (Ed.)

Rewriting Techniques and Applications

20th International Conference, RTA 2009
Brasília, Brazil, June 29–July 1, 2009
Proceedings

Springer

Volume Editor

Ralf Treinen
Université Paris Diderot - Paris 7
Laboratoire PPS
Case 7014, 75205 Paris CEDEX 13, France
E-mail: treinen@pps.jussieu.fr

Library of Congress Control Number: 2009930685

CR Subject Classification (1998): F.4, F.3, D.3.1, I.1, I.2.3, I.2.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-642-02347-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-02347-7 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

springer.com

© Springer-Verlag Berlin Heidelberg 2009
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12696505 06/3180 5 4 3 2 1 0

Preface

This volume contains the papers of the 20th International Conference on Rewriting Techniques and Applications (RTA 2009), which was held from June 29 to July 1, 2009, in Brasília, Brazil as part of the 5th International Conference on Rewriting, Deduction, and Programming (RDP 2009) together with the International Conference on Typed Lambda Calculi and Applications (TLCA 2009), the International School on Rewriting (ISR 2009), the 4th Workshop on Logical and Semantic Frameworks with Applications (LSFA 2009), the 10th International Workshop on Rule-Based Programming (RULE 2009), the 8th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2009), the 9th International Workshop on Reduction Strategies in Rewriting and Programming (WRS 2009), and the annual meeting of the IFIP Working Group 1.6 on term rewriting.

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), and Hagenberg (2008).

For RTA 2009, 22 regular research papers and four system descriptions were accepted out of 59 submissions. Each paper was reviewed by at least three members of the Program Committee, with the help of 94 external reviewers, and an electronic meeting of the Program Committee was held using Andrei Voronkov's EasyChair system. I would like to thank the members of the Program Committee and the external reviewers for their great work, and Andrei Voronkov for providing the EasyChair system, which was invaluable in the reviewing process, the electronic Program Committee meeting, and the preparation of this volume.

The Program Committee decided to split the RTA 2009 best paper award between the paper "An Explicit Framework for Interaction Nets" by Marc de Falco, and the paper "From Outermost to Context-Sensitive Rewriting" by Jörg Endrullis and Dimitri Hendriks.

In addition to the contributed papers, the RTA program contained two invited talks by Vincent Danos and Johannes Waldmann, and jointly with TLCA 2009 an invited talk by Robert Harper.

Many people helped to make RTA 2009 a success. I would like to thank in particular the Conference Chair Mauricio Ayala Rincón, the RTA Publicity Chair Hitoshi Ohsaki, and the local organization team, as well as the sponsors Universidade de Brasília, Brazilian Counsel of Technological and Scientific Development - CNPq, Brazilian Coordination for the Improvement Higher Education Personnel - CAPES, and Federal District Research Foundation - FAPDF.

April 2009 Ralf Treinen

This page intentionally left blank

Organization

Conference Chair

Mauricio Ayala Rincón	Brasília, Brazil

Program Chair

Ralf Treinen	Paris, France

Program Committee

Takahito Aoto	Sendai, Japan
Franz Baader	Dresden, Germany
Eduardo Bonelli	Buenos Aires, Argentina
Dan Dougherty	Worcester, USA
Rachid Echahed	Grenoble, France
Santiago Escobar	Valencia, Spain
Neil Ghani	Glasgow, UK
Jürgen Giesl	Aachen, Germany
Jean Goubault-Larrecq	Cachan, France
Aart Middeldorp	Innsbruck, Austria
Hitoshi Ohsaki	Osaka, Japan
Vincent van Oostrom	Utrecht, The Netherlands
Elaine Pimentel	Belo Horizonte, Brazil
Femke van Raamsdonk	Amsterdam, The Netherlands
Manfred Schmidt-Schauß	Frankfurt, Germany
Sophie Tison	Lille, France
Ashish Tiwari	Stanford, USA

Local Organization Committee

David Deharbe	Natal, Brazil
Flávio L. C. de Moura	Brasília, Brazil
Hermann Haeusler	Rio de Janeiro, Brazil
Elaine Pimentel	Belo Horizonte, Brazil
Alejandro Ríos	Buenos Aires, Argentina
Alberto Pardo	Montevideo, Uruguay

RTA Steering Committee

Maribel Fernández	London, UK
Bernhard Gramlich	Vienna, Austria
Franz Baader	Dresden, Germany
Hitoshi Ohsaki	Osaka, Japan
Albert Rubio	Barcelona, Spain
Johannes Waldmann	Leipzig, Germany

External Reviewers

Abel, Andreas
Adams, Robin
Adao, Pedro
Alarcon, Beatriz
Alpuente, Maria
Blanqui, Frédéric
Bae, Kyungmin
Balat, Vincent
Boneva, Iovka
Bonfante, Guillaume
Boronat, Artur
Bucciarelli, Antonio
Bursuc, Sergiu
Caron, Anne-Cécile
Cheney, James
Chiba, Yuki
Cirstea, Horatiu
Corradini, Andrea
Csaba Ölveczky, Peter
Delaune, Stéphanie
Diaz-Caro, Alejandro
Emmes, Fabian
Endrullis, Jörg
Faure, Germain
Fernández, Maribel
Fuhs, Carsten
Gabbay, Murdoch
Godoy, Guillem
Grabmayer, Clemens
Gramlich, Bernhard
Grattage, Jonathan
Gutierrez, Raul
Hashimoto, Kenji
Hendriks, Dimitri
Hirokawa, Nao
Hofbauer, Dieter
Johannsen, Jacob
Kesner, Delia
Ketema, Jeroen
Khasidashvili, Zurab
Kikuchi, Kentaro
Kop, Cynthia
Küsters, Ralf
Lescanne, Pierre
Levy, Jordi
Lohrey, Markus
López-Fraguas, Francisco
Lucas, Salvador
Lugiez, Denis
Lynch, Christopher
Mackie, Ian
Marion, Jean-Yves
Martí-Oliet, Narciso
Meadows, Catherine
Morawska, Barbara
Moser, Georg
Nakano, Masahiro
Narendran, Paliath
Niehren, Joachim
Nishida, Naoki
Orejas, Fernando
Otto, Carsten
Otto, Friedrich
Parent-Vigouroux, Catherine
Raymond, Pascal
Riba, Colin
Roşu, Grigore
Rubio, Albert

Rusu, Vlad
Sabel, David
Sakai, Masahiko
Schnabl, Andreas
Schneider-Kamp, Peter
Schwinghammer, Jan
Seki, Hiroyuki
Silva, Alexandra
Simonsen, Jakob Grue
Suzuki, Taro
Takai, Toshinori
Talbot, Jean-Marc
Thiemann, René
Urbain, Xavier
Urban, Christian
Verdejo, Alberto
Verma, Kumar Neeraj
Verma, Rakesh M.
Villaret, Mateu
de Vries, Fer-Jan
de Vrijer, Roel
Waldmann, Johannes
Weirich, Stephanie
Yamada, Toshiyuki
Zakharyaschev, Michael
Zankl, Harald

This page intentionally left blank

Table of Contents

Automatic Termination 1
Johannes Waldmann

Loops under Strategies 17
René Thiemann and Christian Sternagel

Proving Termination of Integer Term Rewriting 32
Carsten Fuhs, Jürgen Giesl, Martin Plücker, Peter Schneider-Kamp, and Stephan Falke

Dependency Pairs and Polynomial Path Orders 48
Martin Avanzini and Georg Moser

Unique Normalization for Shallow TRS 63
Guillem Godoy and Florent Jacquemard

The Existential Fragment of the One-Step Parallel Rewriting Theory . . . 78
Aleksy Schubert

Proving Confluence of Term Rewriting Systems Automatically 93
Takahito Aoto, Junichi Yoshida, and Yoshihito Toyama

A Proof Theoretic Analysis of Intruder Theories 103
Alwen Tiu and Rajeev Goré

Flat and One-Variable Clauses for Single Blind Copying Protocols: The XOR Case 118
Helmut Seidl and Kumar Neeraj Verma

Protocol Security and Algebraic Properties: Decision Results for a Bounded Number of Sessions 133
Sergiu Bursuc and Hubert Comon-Lundh

YAPA: A Generic Tool for Computing Intruder Knowledge 148
Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune

Well-Definedness of Streams by Termination 164
Hans Zantema

Modularity of Convergence in Infinitary Rewriting 179
Stefan Kahrs

A Heterogeneous Pushout Approach to Term-Graph Transformation 194
Dominique Duval, Rachid Echahed, and Frédéric Prost

An Explicit Framework for Interaction Nets 209
Marc de Falco

Dual Calculus with Inductive and Coinductive Types 224
Daisuke Kimura and Makoto Tatsuta

Comparing Böhm-Like Trees 239
Jeroen Ketema

The Derivational Complexity Induced by the Dependency Pair Method 255
Georg Moser and Andreas Schnabl

Local Termination 270
Jörg Endrullis, Roel de Vrijer, and Johannes Waldmann

VMTL–A Modular Termination Laboratory 285
Felix Schernhammer and Bernhard Gramlich

Tyrolean Termination Tool 2 295
Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp

From Outermost to Context-Sensitive Rewriting 305
Jörg Endrullis and Dimitri Hendriks

A Fully Abstract Semantics for Constructor Systems 320
Francisco Javier López-Fraguas, Juan Rodríguez-Hortalá, and Jaime Sánchez-Hernández

The Π^0_2-Completeness of Most of the Properties of Rewriting Systems You Care About (and Productivity) 335
Jakob Grue Simonsen

Unification in the Description Logic $\mathcal{EL}$ 350
Franz Baader and Barbara Morawska

Unification with Singleton Tree Grammars 365
Adrià Gascón, Guillem Godoy, and Manfred Schmidt-Schauß

Unification and Narrowing in Maude 2.4 380
Manuel Clavel, Francisco Durán, Steven Eker, Santiago Escobar, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott

Author Index 391

Automatic Termination

Johannes Waldmann

Hochschule für Technik, Wirtschaft und Kultur (FH) Leipzig
Fakultät IMN, PF 30 11 66, D-04251 Leipzig, Germany

Abstract. We give an overview of applications of weighted finite *automata* to *automatically* prove termination of rewriting. Instances of this approach are: standard and arctic matrix interpretations, and the match bound technique. These methods have been developed in recent years, and they are being used by today's leading automated termination prover software.

1 Introduction

A rewriting system R defines a relation $\to_R$ on terms. Considering this as a model of computation, we are interested in derivations $i \to_R^* o$ from input i to output o. We actually want to obtain some output in finite time, so termination is a natural requirement: there is no infinite derivation starting from any i.

Another important application of rewriting is equational reasoning. Here one is concerned with the equivalence defined by the reachability relation $\leftrightarrow_R^*$. Termination comes into play since one wants to express this equivalence via a confluent and terminating rewriting system R' because this makes the reachability problem decidable. In fact, this can be seen as the historic motivation for developing methods for proving termination of rewriting, see Knuth-Bendix completion [24].

In recent years, there are increased efforts to prove termination of programs (logic, functional, imperative) via transformation to a rewriting termination problem, or applying methods from rewriting termination to the original problem.

Special focus is on obtaining termination proofs automatically. Such automatic provers then can be built into tools for completion or program analysis.

The present paper reports on termination methods that use weighted finite automata. They are being developed since 2003 (match-bound method) and it was realized only in 2006 (standard matrix method) that they have a uniform automata theoretic explanation. Once this was understood, the arctic matrix method could be derived "automatically" in 2007.

There has always been a strong connection between rewriting and the theory of automata and formal languages. Indeed, formal grammars are rewriting systems (plus a device for intersection with regular languages to remove remains of intermediate derivation steps since only the result is interesting). Much of the classic theory then is concerned with equivalent ways of describing sets of descendants (reachable by applying grammar rules), of which we mention logic (e.g., monadic second order logic with one successor), algebra (language operations, e.g., regular

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 1–16, 2009.
© Springer-Verlag Berlin Heidelberg 2009

expressions), and (finite) automata (pre-images of homomorphisms into (finite) algebras).

We list some connections between automata theory and rewriting (grammars): all the classes of the Chomsky hierarchy are closed w.r.t. intersection with regular languages, and in fact this can be proved, in each case, by constructing a grammar of the appropriate type that represents the intersection. More to the point, some language classes are known to be closed w.r.t. (many-step) rewriting: e.g., the image of a regular language under a monadic rewriting is again regular [6]. This result is proved by representing the language by a finite automaton, and then applying some closure construction.

2 Automata, Rewriting, ... and Termination?

Indeed this leads us near the topic of the paper. The earliest of the methods under consideration here, namely match bounds, was obtained around 2002, when Dieter Hofbauer and the author visited Alfons Geser (then in Hampton/ Virginia), and together were trying to generalize the following well-known observation on solvable positions in the one-person game *Solitaire*. The object of this game is to remove pegs from a board, where one peg jumps over an adjacent peg, thereby removing it. A one-dimensional board is a string over the alphabet $\Sigma = \{O, X\}$, where X denotes a cell occupied by a peg, and O denotes an empty cell. Then a move of the game is an application of a rewrite step w.r.t. the system $S = \{XXO \to OOX, OXX \to XOO\}$. A solitaire position (a string $i \in \Sigma^*$) is *solvable* if there is a sequence of moves that leads to a position with only one peg, that is, to a string o in $L = O^*XO^*$. It is a folklore theorem that the set of solvable positions is a regular language. In other words, the set of (many-step) predecessors of L w.r.t. $\to_S$ is regular. It is not too hard to guess the corresponding regular expression, and verify it by a careful case analysis. Cristopher Moore and David Eppstein, who give the regular expression in [28], add: "[regularity] was already shown in 1991 by Thane Plambeck and appeared as an exercise in a 1974 book [Mathematical Theory of Computation] by Zohar Manna". Exercise III.5.5 in [4] by Jean Berstel requires to show that the congruence generated by "one-sided solitaire" $XXO \to OOX$ is a rational transduction (thus, regularity preserving).

Bala Ravikumar in [30] proved regularity of solvable solitaire positions on two-dimensional boards with fixed height (but unbounded width). He replaced the guessing of the regular expression by a constructive proof of the theorem that "the solitaire rewriting system preserves regularity of languages", and for the proof he introduced the idea of *change heights*. For a derivation in a length (in fact, shape) preserving rewriting system, the change height of a position measures the number of rewrite steps that touch this position. A system is change-bounded if there is a global bound on change heights, independent of the start term and the actual derivation sequence. Ravikumar's theorem is that each change-bounded string rewriting system preserves regularity of languages, and it can be shown that the change bound for the solitaire system is 4. This raised

the natural question: is there a similar method that works for systems that are not length-preserving?

Note that at this point, the question was to give a constructive proof that certain rewriting systems preserve regularity of languages. From the proof, one could also infer that the rewriting system is terminating, but this was a side effect. In particular, for the solitaire system, termination is trivial (in each step, one peg is removed). With the positive answer given by the *match bound* method, the side effect of proving termination soon became the main attraction.

Yet we postpone the discussion of match bounds because we try to structure the paper not historically, but systematically, as follows. In Section 3, we review automata with weights (in some semiring), as a generalization of classical automata (which are weighted in the Boolean semiring), and explain in Section 4 in general terms how they can certify termination of rewriting. In Section 5, we present the "(standard) matrix method" as an instance of automata with weights in $(\mathbb{N}, +, \cdot)$. For easier exposition, we start with string rewriting. The methods apply to term rewriting as well. We need weighted tree automata, introduced in Section 6. Then in Section 7, we show that the method also works for automata with weights from the arctic semiring $(\{-\infty\} \cup \mathbb{N}, \max, +)$. Using another semiring $(\mathbb{N}, \max, \min)$, we explain the match-bound method in Section 8. We then turn to the question of how do we actually find such certificates of termination for given rewrite systems. We discuss the general *constraint solving* approach in Section 9, and an *automata completion* approach that works especially well for match-bounds in Section 10. We review some open problems in Section 11, where we refer to the notion of matrix termination hierarchy. We close the paper with Section 12 by showing that weighted automata contribute to recent developments on derivational complexity of rewriting.

We do not attempt to give a complete and general overview of methods and techniques in automated termination. We focus on various instances of "matrix methods", by explaining them in the weighted automaton setting. Even there, the presentation will be somewhat biased: we do plan to cover the map, but at the same time emphasize some "forgotten" ideas and point to ongoing work and open problems.

3 Weighted Automata ...

A (classical) finite automaton $A = (\Sigma, Q, I, F, \delta)$ consists of a signature Σ, a set of states Q, sets $I, F \subseteq Q$ of initial and final states, respectively, and a transition relation $\delta \subseteq Q \times \Sigma \times Q$. We imagine the automaton as a directed graph on Q. For each $(p, c, q) \in \delta$, there is an edge from p to q labelled c. There may be loops and parallel edges. A path is a connected sequence of edges, and the label of a path is the word obtained as the concatenation of its edge labels. Write $A(p, w, q)$ for the statement "there is a path with label w from state p to state q". The automaton computes a Boolean-valued function $A : \Sigma^* \to \mathbb{B}$ with $A(w) = \bigvee\{A(p, w, q) \mid p \in I, q \in F\}$. Paths are combined sequentially and in parallel. Along a path, all edges must be present (conjunction); while there may be several paths with identical label, one of which is enough (disjunction).

This can be generalized: we can replace the Boolean domain $(\mathbb{B}, \vee, \wedge)$ with any other suitable structure. Then, an edge (still labelled with a letter) is not just present or absent, but it carries a weight, where weights are taken from a *semiring*. The semiring provides operations of addition (used in parallel composition) and multiplication (for sequential composition), and fulfill several axioms that just map naturally to the idea of composing paths and their weights.

Formally, a W-weighted automaton A consists of $(\Sigma, Q, I, F, \delta)$ where function $\delta \subseteq Q \times \Sigma \times Q \to W$ assigns weights to transitions, and the sets of initial (final, resp.) states are replaced by initial (final, resp.) weight assignments I (F, resp.) Each path from p to q with label u now has a weight $A(p, u, q) \in W$, computed as the product of its edge weights. The automaton assigns to a word $u \in \Sigma^*$ the value $A(u) = \sum\{I(p) \cdot A(p, u, q) \cdot F(q) \mid p, q \in Q\}$.

It is a nice coincidence that the talk will be given in Brazil: much of today's knowledge on weighted automata builds on work by Brazilian scientist Imre Simon (`http://www.ime.usp.br/~is/`) who used finite automata over the $(\mathbb{N}, \min, +)$ semiring in a decision procedure for the Finite Power Property of regular languages [31], and other problems in formal languages (for an more recent overview, see [32]). In fact this semiring was later named with reference to him the *tropical semiring*. We use its counterpart, the arctic semiring, in Section 7.

4 . . . for Termination of Rewriting

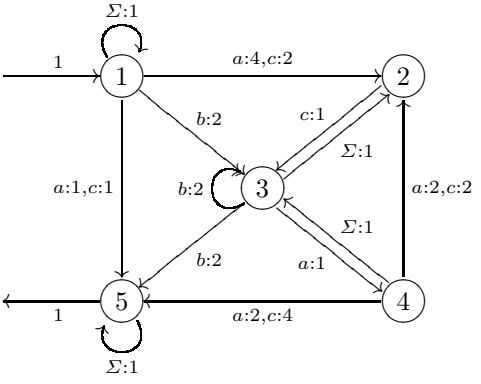
If the weight semiring W of the automaton A is well-founded w.r.t. some order $>$, and A fulfills the following condition w.r.t. a rewriting system R, called *global compatibility*: $u \to_R v$ implies $A(u) > A(v)$, then R is terminating. This is a trivial statement and it is not effective since in general there seems to be no way to test global compatibility, since we need to check this condition for all terms and rewrite steps. We turn this approach into a useful termination method by giving a *local compatibility* condition on the automaton that implies global compatibility and that can be checked effectively, e.g., by inspection of a finite number of cases. Roughly speaking, the infinite number of rewrite steps of the system R is partitioned into a finite number of classes by considering all possible locations of the rewrite step "in the automaton".

We will strive to formulate local compatibility in such a way that for a given rewrite system, an automaton that fulfill the conditions can (in principle) be found by automated constraint solver software. Such an automaton is then indeed a (finite) certificate for termination. Its validity can be checked independently from the way it was obtained. One application is automated verification of termination proofs, which is done in a two-step process: the underlying theorems (local compatibility $\Rightarrow$ global compatibility $\Rightarrow$ termination) for standard [26] and arctic [25] matrix interpretations have been formalized and proved by Adam Koprowski and are now part of the Color library of certified termination techniques [5]. Then for each concrete termination certificate, the correct application of the theorem has to be checked. With our notions of local compatibility, this is (conceptually) easy, since it consists of checking the solution of a constraint system, see Section 9.

5 Matrix Interpretations

As a first instance of the general scheme, we describe how automata with weights in the "standard" semiring of natural numbers $(\mathbb{N}, +, \cdot)$ are used for proving termination of string rewriting. (For term rewriting, see Section 6.)

Example 1. This is the automaton from [21] that helped to sell the matrix method, because it certifies termination of $Z_{086} = \{a^2 \to bc, b^2 \to ac, c^2 \to ab\}$, which solved an open problem. (Here, Z_{086} is the name of this problem in the Termination Problem Data Base (TPDB), where "Z" refers to its author Hans Zantema.)



For all pairs (p, q) of states, and rules $(l \to r) \in R$, we consider the weights $A(p, l, q)$ and $A(p, r, q)$ computed by the automaton. We require weak local compatibility everywhere: $A(p, l, q) \geq A(p, r, q)$, this already ensures that $u \to_R v$ implies $A(u) \geq A(v)$. In the example, we have,e.g., $2 = A(4, bb, 3) \geq A(4, ac, 3) = 2$. □

Additionally, we need a strict decrease in a few well-chosen places. The original matrix paper [21] treated this with a theory based on positive cones in rings, which we here present in an automata theoretic setting:

Proposition 2. *For any string rewriting system $R = \{l_1 \to r_1, \ldots, l_n \to r_n\}$ over Σ, and $(\mathbb{N}, +, \cdot)$-weighted automaton $A = (\Sigma, Q, I, F, \delta)$ that is weakly locally compatible with R, define the language $D \subseteq \Sigma^* \cdot \{1, \ldots, n\} \cdot \Sigma^*$ as*

$$\left\{ u \cdot k \cdot v \mid \exists i, p, q, f \in Q : \begin{array}{lll} I(i) \geq 1 \wedge & A(i, u, p) \geq 1 & \\ \wedge\, A(p, l_k, q) > A(p, r_k, q) & & \\ \wedge & A(q, v, f) \geq 1 & \wedge F(f) \geq 1 \end{array} \right\}.$$

If $D = \Sigma^* \cdot \{1, \ldots, n\} \cdot \Sigma^*$, then A is globally compatible with R. This condition on D is decidable. □

The language D encodes the set of all reachable and productive redex/contractum positions for which the automaton computes a decrease, so the first part of the statement is immediate. The second part is seen as follows: if we map 0

to "false", and each positive number to "true", then we have a semiring morphism from $(\mathbb{N}, +, \cdot)$ to $(\mathbb{B}, \vee, \wedge)$ that also respects the ordering (where "false" $<$ "true"). This means the languages $\{u \mid A(I, u, p) > 0\}$ and $\{v \mid A(q, v, F) > 0\}$ are effectively regular. The triplets $T = \{(p, k, q) \mid A(p, l_k, q) > A(p, r_k, q)\}$ can be found by considering finitely many cases, so the condition can be decided with methods from classical automata theory. We remark that this involves (roughly) a check whether a non-deterministic finite automaton accepts Σ^*, which is expensive (in fact, PSPACE-complete).

Example 1 meets the conditions of the above proposition, since $\{(1, k, 5) \mid 1 \leq k \leq 3\} \subseteq T$, e.g., $4 = A(1, bb, 5) > A(1, ac, 5) = 2$, and for each $u, v \in \Sigma^*$ we have $A(1, u, 1) > 0$ and $A(5, v, 5) > 0$ because of the loops in the respective states. We remark that the paper [21] contains other realizations of strict local compatibility, but none of them seems to have been used seriously in implementations.

If our goal is proving top termination, e.g., because we applied the dependency pairs transformation [2], then the above scheme can be adapted easily by restricting $u = \epsilon$ in Proposition 2.

As described in work with Andreas Gebhardt and Dieter Hofbauer [14], these methods work even in larger semirings like the non-negative rational, algebraic or real numbers. The natural order $>$ on those domains is not well-founded, so we replace it by $x >_\epsilon y \iff x > \epsilon + y$, for some fixed $\epsilon > 0$. The nice thing is that we do not need to change Proposition 2, since we carefully wrote ≥ 1 which is equivalent to > 0 on the naturals, but does the right thing on the dense domains. Note that we can take

$$\epsilon = \min\{d \mid (l \to r) \in R, d = A(p, l, q) - A(p, r, q), d > 0\}$$

which is positive since R is finite.

6 Weighted Tree Automata

For easier exposition, so far we only considered string rewriting. Since most "real" data is (tree-)structured, there is some interest in term rewriting, and we show how our methods generalize. We need the concept of weighted tree automaton [10,9]. This is a finite state device that computes a mapping from trees over some signature into some semiring. This computational model is obtained from classical (Boolean) tree automata by assigning weights to transitions.

Formally, a W-weighted tree automaton is a tuple $A = (\Sigma, Q, \delta, F)$ where W is a semiring, Q is a finite set of states, Σ is a ranked signature, δ is a transition function that assigns to any k-ary symbol $f \in \Sigma_k$ a function $\delta_f : Q^k \times Q \to W$ and F is a mapping $Q \to W$. The idea is that $\delta_f(q_1, \ldots, q_k, q)$ gives the weight of the transition from $(q_1, \ldots, q_k)$ to q, and $F(q)$ gives the weight of the final state q.

The semantics of a weighted tree automaton is defined as follows: a run of A on a tree t is a mapping from positions of t to states of A, the weight of a run is the product of the weights of its transitions, and the weight of the term is the sum of the weight of all its runs. We emphasize here another, equivalent

approach: the automaton is a Σ-algebra where the carrier set consists of weight vectors, indexed by states. Let $V = (Q \to W)$ be the set of such vectors. Then for each k-ary symbol f, the transition δ_f computes a function $[\delta_f] : V^k \to V$ by $[\delta_f](\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k) = \boldsymbol{w}$ where

$$\boldsymbol{w}_q = \sum\{\delta_f(q_1, \ldots, q_k, q) \cdot \boldsymbol{v}_{1,q_1} \cdot \ldots \cdot \boldsymbol{v}_{k,q_k} \mid q_1, \ldots, q_k \in Q\}.$$

A weighted tree automaton realizes a multilinear algebra: each function $[\delta_f]$ is linear in each argument.

In order to use tree automata for automated termination, the local compatibility condition should be easy. Therefore we only consider automata whose transition functions can be written as sums of unary linear functions. These are matrices, so we arrive at the notion of matrix interpretation, using functions $V^k \to V$ of shape

$$(\boldsymbol{v_1}, \ldots, \boldsymbol{v_k}) \mapsto M_1 \cdot \boldsymbol{v_1} + \ldots + M_k \cdot \boldsymbol{v_k} + \boldsymbol{a}, \tag{1}$$

where each M_i is a square matrix, and $\boldsymbol{a}$ is a vector, and all vectors are column vectors.

The corresponding tree automata are called *path-separated* because their semantics can be computed as the sum of matrix products along all paths of the input tree, and the values along different paths do not influence each other.

With these preparations, we can apply the monotone algebra approach [13] for proving termination of term rewriting, where the algebra is given by a path-separated weighted tree automaton. The order on the vector domain depends on the chosen weight semiring, and on the question whether we want closure under contexts (we don't need this for top rewriting). In each case, we can take some modification of the pointwise extension of the semiring order.

We briefly discuss the relation to polynomial interpretations [7], which are a well-known previous instance of monotone algebras for termination. The distinctive features are that polynomial interpretations can be non-linear while matrix interpretations are linear, but this is complemented by the fact that polynomial interpretation use a totally ordered domain (of natural or real numbers) while the domain for matrix interpretations consists of vectors with the point-wise ordering, which is non-total.

7 Half-Strict Semirings

The formulation of a sufficient local compatibility condition depends on properties of the semiring operations. We call an operation $\circ$ *strict* w.r.t. an order $>$ if $x_1 > x_2$ implies $x_1 \circ y > x_2 \circ y$. On naturals, standard addition is strict, and standard multiplication is strict for $y \neq 0$.

We intend to use the arctic semiring $(\{-\infty\} \cup \mathbb{N}, \max, +)$, and we immediately notice that the "max" operation is not strict. Still it does have the following property: $x_1 > x_2 \wedge y_1 > y_2$ implies $\max(x_1, y_1) > \max(x_2, y_2)$. We call this

half strict, since we need two strict decreases in the arguments to get one strict decrease in the result.

Now look at one strict decrease for a redex, $A(p,l,q) > A(p,r,q)$, in some weighted string automaton A. Strict multiplication produces from that a strict decrease when we apply a context (left and right). The automaton adds the weight of all paths with identical label. If addition is strict, then *one* decreasing path is enough, and that was the idea behind Proposition 2.

Now in the arctic semiring, addition is half-strict, so a sufficient condition for a global weight decrease is that *all* redex paths must be decreasing. We can make an exception for redex paths of weight zero (i.e., "missing" paths), since the max operation is strict when one argument is $-\infty$. Therefore we compare arctic weights by $x \gg y \iff x > y \vee x = y = -\infty$, and arctic vectors by the pointwise extension of that. Since $\gg$ is not well-founded, we need to make sure that we never produce a total weight vector $-\infty^d$. This can be achieved by requiring that the first vector element is "positive" (that is, $> -\infty$). This restricts the domain of the algebra, so the operations have to respect that. It is enough to require that the upper left entry of the matrices is positive as well. We remark here that the focus on the first vector component is just one way of reaching the goal, and the derivation of a more general criterion in the spirit of Proposition 2 is left as an exercise.

Example 3. Again, we prove termination of $Z_{086} = \{a^2 \to bc, b^2 \to ac, c^2 \to ab\}$. The nontrivial dependency pairs are $\{Aa \to Bc, Bb \to Ac\}$. Take the arctic two-state automaton with transition matrices

$$[a] = \begin{pmatrix} 0 & 3 \\ 2 & 1 \end{pmatrix}, [b] = \begin{pmatrix} 3 & 2 \\ 1 & -\infty \end{pmatrix}, [c] = \begin{pmatrix} 0 & 1 \\ 3 & 2 \end{pmatrix}, [A] = [B] = (0 \ -\infty),$$

where we use a reduced matrix shape for the top symbols (only the transitions from the initial state). Then we have these weak compatibilities

$$[a^2] = \begin{pmatrix} 5 & 4 \\ 3 & 5 \end{pmatrix} \geq \begin{pmatrix} 5 & 4 \\ 1 & 2 \end{pmatrix} = [bc], \qquad [b^2] = \begin{pmatrix} 6 & 5 \\ 4 & 3 \end{pmatrix} = [ac]$$

$$[c^2] = \begin{pmatrix} 4 & 3 \\ 5 & 4 \end{pmatrix} \geq \begin{pmatrix} 4 & 2 \\ 5 & 4 \end{pmatrix} = [ab], \qquad [Aa] = (0\ 3) \geq (0\ 1) = [Bc]$$

and the strict compatibility $[Bb] = (3\ 2) > (0\ 1) = [Ac]$. This allows to remove one rule and the rest is trivial (counting symbols). □

Another hard termination problem was $\{b^3 \to a^3, a^3 \to aba\}$. Termination could not be established automatically by any of the programs (nor their authors) taking part in the competition 2006. Then, Aleksey Nogin and Carl Witty produced a handwritten proof, that was later generalized to the method of *quasi-periodic interpretations* [35]. It is now known that quasi-periodic interpretations of slope one over unary signatures can be translated into arctic matrix interpretations (see full version of [25], submitted). In fact, Matchbox used this translation in the termination competition of 2008.

We now turn to arctic interpretations for proving termination of term rewriting. We restrict to path-separated automata, as described earlier. At each position of a tree, values from subtrees are added. The redex position might be in any of the subtrees, and this creates a problem: we want then a decrease of the value of the tree, but arctic addition is not strict. This rules out the possibility of using path-separated arctic tree automata for proving full termination. They are still useful: the subtree problem does not appear when there are no redexes in subtrees, and this happens exactly when the redex position is in the root. So, arctic tree automata are applicable for proving top termination. (Of course there are non-top redexes but we don't need a strict decrease when reducing them.) This plan has been carried out and it is described in [25]. This paper also contains an extension to arctic numbers "below zero", i.e., the semiring $(\{-\infty\} \cup \mathbb{Z}, \max, +)$. Restricting the first component of the vectors to be positive works again.

8 Match Heights

As mentioned in the introduction, the idea of annotating positions in strings by numbers that give some indication of their rewrite history, derives from Ravikumar's concept of change bounds. The restriction to length-preserving rewriting can be dropped by considering match-heights instead. Here, the match height in the contractum is 1 larger than the lowest match-height in the redex. It is proved in [16] that match-bounded string rewriting systems are terminating, and effectively preserve regularity of languages.

The relation to weighted automata was explained only some time later [33], and uses the $(\mathbb{N}, \min, \max)$ semiring. Here, none of the semiring operations is strict, so we expect complications. And indeed, both semiring addition and semiring multiplication are idempotent, and that means that for any finite weighted automaton A, the range $A(\Sigma^*)$ of the automaton's semantics function is finite. This also bounds the length of decreasing chains of weights by a constant, and that in turn bounds the length of derivations of the rewriting system we hoped the automaton to be strictly compatible with. This seems to rule out the use of this semiring completely, since every non-empty string rewriting system has at least linear, thus unbounded derivation lengths.

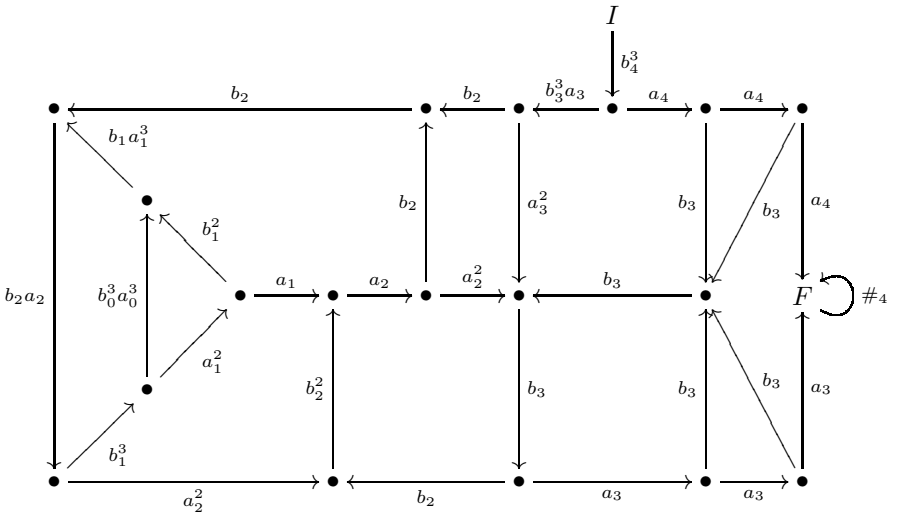
The solution is: we still apply a local compatibility condition in $(\mathbb{N}, \min, \max)$, but globally, we use a different semiring (where multiplication is not idempotent): its elements are the finite multisets of $\mathbb{N}$, semiring multiplication is multiset union, and semiring addition is the "min" operation w.r.t. the lexicographic ordering. Note that the multiplicative unit is the empty multiset, and for the additive unit we introduce an extra element ∞, larger than all others and absorbing for multiplication.

Now each $(\mathbb{N}, \min, \max)$-automaton can be lifted to an automaton in this multiset semiring, by changing each weight w into the multiset weight $\{w\}$, and the value that the automaton computes for some word u is the smallest (in the above sense) multiset of edge weights of a path labelled u.

The strict local compatibility condition (for the automaton in the original semiring) reads as expected: we require that for all $(l \to r) \in R, p, q \in Q$:

$A(p,l,q) \gg A(p,r,q)$ where $x \gg y$ if $x > y \vee x = y = +\infty$. Basically, we replace the lexicographic comparison of multisets by the comparison of their respective maximum elements. That is, we ignore multiplicities.

Example 4. The following automaton is strictly locally compatible with Zantema's system $Z_{001} = \{a^2b^2 \to b^3a^3\}$, and the picture (drawn by Dieter Hofbauer) is too nice to be omitted here.



We use "edge compression", e.g., $\bullet \overset{a_2^2}{\to} \bullet$ really means $\bullet \overset{a_2}{\to} \bullet \overset{a_2}{\to} \bullet$ containing one additional state. The indices on the letters indicate weights. These are from the semiring $(\mathbb{N} \cup \{\infty\}, \min, \max)$, since all "missing edges" have weight ∞. The symbol # is referring to the RFC method, see Section 10. The given automaton constituted (in 2003) the first automated termination proof for Z_{001}, while only "hand crafted" proofs were available [34]. □

The above presentation may not look like the standard version of match-bounded termination, but note that already there we used a multiset argument to bound lengths of derivations. The approach given here leads to extensions of the standard method, e.g., for relative termination. Note that the obvious $A(p,l,q) \geq A(p,r,q)$ for weak local compatibility in the original automaton does not imply weak compatibility in the lifted automaton. But this can be repaired by replacing $\geq$ with $\gg'$ where $x \gg' y$ if $x > y \vee x = y = +\infty \vee x = y = -\infty$. That is, the "relative rules" must behave like strict rules (decrease weights) except they may keep the lowest weight $(-\infty)$.

Is this useful for proving termination (without relative rules)? It is easy to show that a sequence of rule removals by relative match-bound proofs still implies a match-bound on the original system. Even so, the sequence of relative proofs may be easier to find automatically.

The idea of termination proofs via height annotations has been applied in term rewriting. There, one uses $(\mathbb{N}, \min, \max)$-weighted tree automata, again with a suitable local compatibility condition, where one has to take into account the position of variables in rewrite rules, resulting in the concept of roof-bounds [19].

9 Constraint Solving

One method of finding matrix interpretations (weighted automata) is constraint programming: write down all the properties, and find a satisfying assignment by some constraint solver software. At top level, the unknowns are the coefficients of linear functions with the shape of Equation 1, and the constraints relate linear functions that represent interpretations of left-hand sides and right-hand sides of rewrite rules. Here, it comes in handy that these functions are closed under substitution. This is caused by path-separation and would not hold for general automata. Constraints for linear functions can be translated to constraints on matrices; and constraints for matrices can be translated to constraints on their elements. Here, we arrived at constraints for natural numbers, since all our semirings use numbers.

Numbers can be represented in binary notation, and numerical constraints should be formulated in the SMT (satisfiability modulo theories) language QB-BV (quantifier free bit vector arithmetics). Instead, the numerical constraint is translated into CNF-SAT and then a SAT solver is applied. It seems strange that "manual" translation of termination constraints to CNF-SAT should be more efficient than applying QF-BV solvers. One explanation could be that so far, the power of matrix termination provers is increased by taking larger matrices, rather than larger matrix entries. But this may be a self-fulfilling prophecy.

The constraint solver implementation in Matchbox in competition 2006 was internally dubbed "the one-bit wonder" since it used a bit width of *one* only. Today, Matchbox is using carefully optimized CNF-SAT encodings for binary arithmetics on standard and arctic naturals of bit widths 3 and 4. These were computed with the help of Peter Lietz. The translation of the Z_{001} and Z_{086} constraint systems for matrix dimension 4 and bit width 3 produces ≈ 3000 boolean variables and ≈ 20.000 clauses, of which Minisat [11] eliminates just one percent and then solves them in a few seconds.

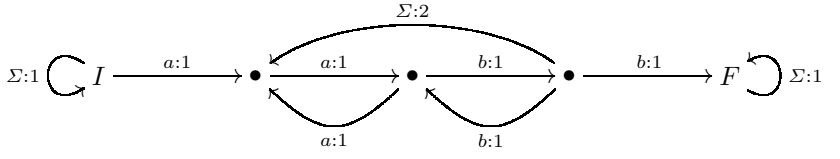
It is an interesting idea to look for $(\mathbb{N}, \min, \max)$ interpretations via constraint solving. This may prove useful in connection with relative match-bound methods, see the discussion at the end of Section 8. E.g., we can prove that the TPDB systems SRS/Zantema06/$\{15, \ldots, 18\}$ are match-bounded. (This seems out of reach of completion methods.) Why would we want to do this? We could prove termination by other methods, but match-boundedness gives linear complexity, see Section 12.

10 Automata Completion

The constraint system that describes local compatibility requires certain inequalities between weights of paths in the automaton. We can fix the number of

states, and then find appropriate edge weights, as described in the previous section. Another idea is to start with some candidate automaton, and then extend it by adding transitions *and states*, until all constraints are met. The problem is that additional states lead to additional constraints, and it is not clear how to organize this into a terminating algorithm.

Example 5. Dieter Hofbauer's termination prover MultumNonMulta contains a version of weighted automata completion for the standard semiring $(\mathbb{N}, +, \cdot)$. It finds the following beautiful proof for (again) Z_{001} by starting with the redex path from left to right, and then adding three back edges.



□

We discuss $(\mathbb{N} \cup \{\infty\}, \min, \max)$ now which we need for match-bounds. This semiring has the interesting property that its zero element (∞, which is neutral for the "min" operation) is maximal in the natural ordering. Consider local compatibility constraints $A(p, l, q) \gg A(p, r, q)$. We claim they can be read as "for every non-zero redex, there must be a smaller contractum". Indeed for zero-weight redexes, the constraint is true by the definition of $\gg$.

For a weighted automaton A, denote by $\mathrm{supp}(A)$ the set of words that get non-zero weight. If the weight semiring has zero as its maximum element, then local compatibility implies that $\mathrm{supp}(A)$ is closed under rewriting. Thus if we know that $L \subseteq \mathrm{supp}(A)$ and A is strictly locally compatible with R, then R is terminating on L. In other words, we have a method for proving *local termination* [8]. This allows to do termination proofs by considering right hand sides of forward closures [16]. They can be computed by the rewriting system

$$R' = R \cup \{l_1 \# \to r \mid (l_1 l_2 \to r) \in R, l_1 \neq \epsilon \neq l_2\},$$

starting with $L = \mathrm{rhs}(R)\#^*$, cf. Example 4.

Now, how do we realize the completion of automata? The basic approach is that whenever $A(p, l, q)$ is nonzero (= present), but $A(p, r, q)$ is zero (= missing), then we add a fresh path from p to q, labelled r and weighted appropriately. Several completion results for classical automata are available from the literature. In simple cases, they prove that the basic approach terminates. This happens, e.g., for monadic systems, where $|r| \leq 1$ and thus we never add states, only edges, and we will obtain a saturated automaton.

For rules with $|r| > 2$, this will not work. We should then employ some heuristics that tries to avoid the creation of too many fresh states [18]. A similar idea can be applied to tree automata [19] but in its basic version, it only works for left-linear rewriting. Non-linearities can be handled with quasi-deterministic automata as described by Martin Korp and Aart Middeldorp in

[27]. The methods are sound but not complete (they may fail to find compatible automata). We now describe a complete method for string rewriting.

Instead of a W-weighted automaton A over Σ we can also consider a classical (Boolean) automaton B over $\Sigma \times W$. Local compatibility of A w.r.t. a rewriting system R then translates into local compatibility of B w.r.t. an annotated rewriting system R_W over $\Sigma \times W$. For $W = (\mathbb{N}, \min, \max)$, this annotated system contains rules $l_W \to r_W$ such that the maximal annotation in the left is larger than the maximal annotation in the right. In other words, a rewrite step deletes the maximal letter. We have an instance of a *deleting* string rewriting system [20]: there is a well-founded order on the alphabet such that in reach rule $l \to r$, there is one letter in l that is larger than each letter in r. Deleting systems are terminating and preserve regularity of languages. The idea behind the proof is that for a deleting system R, the rewrite relation $\to_R^*$ can be represented as the composition of two rewrite relations $\to_C^* \circ \to_E^*$ over an auxiliary alphabet, where C is SN $\cap$ CF (not the French railway, but Strongly Normalizing and Context-Free: left-hand sides have length 1) and E is inverse context-free (right-hand sides have length ≤ 1).

The proof is constructive and indeed it was realized in the termination prover Matchbox (2003). However this implementation was soon outperformed by Hans Zantema's implementation of a completion heuristics in Torpa (2004). The situation was reversed again in 2006 when Jörg Endrullis found a substantial improvement for handling deleting systems [12] and implemented it for Jambox: the auxiliary alphabet can be small and the closures w.r.t. C and E can be computed in an interleaved manner. This gives us the best of both worlds: the construction is fast (it can build automata of $> 10^4$ states in < 10 seconds) and it is complete (if a match-bound certificate exists, it will be found). We remark that this seems to be the only instance of an weighted automata method for termination where we have a complete construction. This result also implies that the question "is a given rewriting R system match-bounded by k" is decidable. When k is not given, decidability remains open.

If we reverse all arrows in a (C, E) decomposition of a deleting system, we get some results on "inverse match-bounded" rewriting [17]. There should be a connection to the $(\mathbb{N}, \max, \min)$ semiring, but it is not immediate, e.g., the above multiset argument does not work, and indeed inverse match-boundedness does not imply termination, but termination is decidable. We observed [22] that match-boundedness, inverse match-boundedness, and change-boundedness are equivalent for length-preserving string rewriting systems. In particular this holds for the Solitaire rewriting system.

11 Matrix Termination Hierarchy

Typically, a proof of termination is obtained by a sequence of rule removals. In [15], Andreas Gebhardt and the author propose the notation $R \vdash S$ for rewriting systems R, S with $R \supseteq S$ and $(R \setminus S)$ is terminating relative to S, that is, from R, all non-S rules "could be removed". The relation $\vdash$ is indeed transitive, and

$R \vdash^* \emptyset$ implies termination of R. Let $\mathfrak{M}(W, n)$ be the set of pairs of rewriting systems (R, S) such that if there is a W-weighted automaton with $\leq n$ states that is strictly compatible with $R \setminus S$ and weakly compatible with S. Thus $(R, S) \in \mathfrak{M}(W, n)$ implies $R \vdash S$ and therefore we also write $R \vdash^{\mathfrak{M}(W,n)} S$.

Since $\mathfrak{M}(W, n)$ is a relation on rewriting systems, we make use of standard operations on relations like composition, iteration (exponentiation) and (reflexive and) transitive closure. Then, the collection $\mathfrak{M}(W, n)^s$ is the "matrix termination hierarchy". We have some obvious (non-strict) inclusions for the levels of this hierarchy, considering the embeddings of natural $\subset$ rational $\subset$ algebraic $\subset$ real numbers (on the non-negative subset, with standard operations and ordering), and the monotonicity w.r.t. number of states and proof steps. Then, interesting questions can be raised, like, which levels of the hierarchy are inhabited, which are decidable, and which of the obvious inclusions are strict.

The paper [15] investigates weight domains that are sub-semirings of $\mathbb{R}_{\geq 0}$, and contains some concrete results, like $\mathfrak{M}(\mathbb{N}, 0) \subset \mathfrak{M}(\mathbb{N}, 1) \subset \mathfrak{M}(\mathbb{N}, 2) \subset \mathfrak{M}(\mathbb{N}, 3)$, and some general statements. E.g., the Amitsur-Levitski-Theorem [23] implies that the dimension hierarchy is infinite, and investigation of derivation lengths implies that the proof length hierarchy is infinite. Many questions remain open, and we did not even start to investigate this hierarchy for non-strict semirings.

12 Weighted Automata for Derivational Complexity

Recent work of Georg Moser et al. revived the idea of extending termination analysis of rewriting systems towards complexity analysis. Formally, the goal is to bound lengths of derivations by some function of the size of the starting term. Weighted automata methods contribute some results here. The basic idea is that if A is strictly compatible with R, then derivation lengths from t are bounded by the height of $A(t)$ in the order of the semiring.

For standard matrix interpretations, i.e., path-separated $(\mathbb{N}, +, \cdot)$-automata, $A(t)$ is bounded by some exponential function of $\mathrm{depth}(t)$, since each matrix computes a linear function. By restricting the shape of the matrices, this bound can be lowered, and one instance is that upper triangular matrices give a polynomially bounded interpretation [29].

When we change the semiring, we get even lower bounds: arctic matrix interpretations imply a linear bound, and $(\mathbb{N}, \min, \max)$ interpretations do as well. This holds even without any restrictions on the matrix shapes. In earlier work, the space complexity of computations was bounded by max/plus polynomial quasi-interpretations [1].

One challenge is to lift the "upper triangular" shape restriction for standard matrix interpretations, and still get polynomial bounds. A nice test case is Z_{086}. It is widely believed that this system has quadratic derivational complexity, but it has resisted all attempts to prove this. The interpretation computed by the automaton in Example 1 is not polynomially bounded, as it contains a cycle with weight > 1 (in state 3). Perhaps the arctic automaton in Example 3 helps?

References

1. Amadio, R.M.: Synthesis of max-plus quasi-interpretations. Fundam. Inform. 65(1-2), 29–60 (2005)
2. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoretical Computer Science 236(1-2), 133–178 (2000)
3. Baader, F. (ed.): RTA 2007. LNCS, vol. 4533. Springer, Heidelberg (2007)
4. Berstel, J.: Transductions and Context-Free Languages. Teubner, Stuttgart (1979)
5. Blanqui, F., Coupet-Grimal, S., Delobel, W., Hinderer, S., Koprowski, A.: CoLoR, a Coq library on rewriting and termination. In: Workshop on Termination (2006), `http://color.loria.fr`
6. Book, R.V., Jantzen, M., Wrathall, C.: Monadic thue systems. Theor. Comput. Sci. 19, 231–251 (1982)
7. Cherifa, A.B., Lescanne, P.: Termination of rewriting systems by polynomial interpretations and its implementation. Sci. Comput. Program. 9(2), 137–159 (1987)
8. de Vrijer, R., Endrullis, J., Waldmann, J.: Local termination. In: Treinen, R. (ed.) RTA 2009. LNCS, vol. 5595, pp. 270–284. Springer, Heidelberg (2009)
9. Droste, M., Pech, C., Vogler, H.: A kleene theorem for weighted tree automata. Theory Comput. Syst. 38(1), 1–38 (2005)
10. Droste, M., Vogler, H.: Weighted tree automata and weighted logics. Theor. Comput. Sci. 366(3), 228–247 (2006)
11. Eén, N., Biere, A.: Effective preprocessing in sat through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005)
12. Endrullis, J., Hofbauer, D., Waldmann, J.: Decomposing terminating rewriting relations. In: Workshop on Termination (2006)
13. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of term rewriting. J. Autom. Reasoning 40(2-3), 195–220 (2008)
14. Gebhardt, A., Hofbauer, D., Waldmann, J.: Matrix evolutions. In: Hofbauer, D., Serebrenik, A. (eds.) Proc. Workshop on Termination, Paris (2007)
15. Gebhardt, A., Waldmann, J.: Weighted automata define a hierarchy of terminating string rewriting systems. In: Droste, M., Vogler, H. (eds.) Proc. Weighted Automata Theory and Applications, Dresden, pp. 34–35 (2008)
16. Geser, A., Hofbauer, D., Waldmann, J.: Match-bounded string rewriting systems. Appl. Algebra Eng. Commun. Comput. 15(3-4), 149–171 (2004)
17. Geser, A., Hofbauer, D., Waldmann, J.: Termination proofs for string rewriting systems via inverse match-bounds. J. Autom. Reasoning 34(4), 365–385 (2005)
18. Geser, A., Hofbauer, D., Waldmann, J., Zantema, H.: Finding finite automata that certify termination of string rewriting systems. Int. J. Found. Comput. Sci. 16(3), 471–486 (2005)
19. Geser, A., Hofbauer, D., Waldmann, J., Zantema, H.: On tree automata that certify termination of left-linear term rewriting systems. Inf. Comput. 205(4), 512–534 (2007)
20. Hofbauer, D., Waldmann, J.: Deleting string rewriting systems preserve regularity. Theor. Comput. Sci. 327(3), 301–317 (2004)
21. Hofbauer, D., Waldmann, J.: Termination of string rewriting with matrix interpretations. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 328–342. Springer, Heidelberg (2006)

22. Hofbauer, D., Waldmann, J.: Equivalence of match-boundedness, change-boundedness and inverse match-boundedness for length-preserving string rewriting. Theorietag der FG Automaten und Sprachen der GI, Leipzig (2007), `http://www.imn.htwk-leipzig.de/~waldmann/talk/07/tt/`
23. Kanel-Belov, A., Rowen, L.H.: Computational Aspects of Polynomial Identities. AK Peters (2005)
24. Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebra. In: Proc. Conf. Computational Problems in Abstract Algebra, pp. 263–297. Pergamon Press, Oxford (1967)
25. Koprowski, A., Waldmann, J.: Arctic Termination ...Below zero. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 202–216. Springer, Heidelberg (2008)
26. Koprowski, A., Zantema, H.: Certification of proving termination of term rewriting by matrix interpretations. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 328–339. Springer, Heidelberg (2008)
27. Korp, M., Middeldorp, A.: Proving termination of rewrite systems using bounds. In: Baader [3], pp. 273–287 (2007)
28. Moore, C., Eppstein, D.: One-dimensional peg solitaire, and duotaire. CoRR, math.CO/0008172 (2000)
29. Moser, G., Schnabl, A., Waldmann, J.: Complexity analysis of term rewriting based on matrix and context dependent interpretations. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany. Dagstuhl Seminar Proceedings, vol. 08004 (2008)
30. Ravikumar, B.: Peg-solitaire, string rewriting systems and finite automata. Theor. Comput. Sci. 321(2-3), 383–394 (2004)
31. Simon, I.: Limited subsets of a free monoid. In: FOCS, pp. 143–150. IEEE, Los Alamitos (1978)
32. Simon, I.: On semigroups of matrices over the tropical semiring. ITA 28(3-4), 277–294 (1994)
33. Waldmann, J.: Weighted automata for proving termination of string rewriting. Journal of Automata, Languages and Combinatorics 12(4), 545–570 (2007)
34. Zantema, H., Geser, A.: A complete characterization of termination of $0^p\ 1^q \to 1^r\ 0^s$. Appl. Algebra Eng. Commun. Comput. 11(1), 1–25 (2000)
35. Zantema, H., Waldmann, J.: Termination by quasi-periodic interpretations. In: Baader [3], pp. 404–418 (2007)

Loops under Strategies

René Thiemann and Christian Sternagel*

Institute of Computer Science, University of Innsbruck, Austria
{rene.thiemann,christian.sternagel}@uibk.ac.at

Abstract. Most techniques to automatically disprove termination of term rewrite systems search for a loop. Whereas a loop implies non-termination for full rewriting, this is not necessarily the case if one considers rewriting under strategies. Therefore, in this paper we first generalize the notion of a loop to a loop under a given strategy. In a second step we present two novel decision procedures to check whether a given loop is a context-sensitive or an outermost loop. We implemented and successfully evaluated our method in the termination prover T$_\text{T}$T$_2$.

1 Introduction

Termination is an important property of term rewrite systems (TRSs). Therefore, much effort has been spent on developing and automating powerful techniques for showing termination of TRSs. An important application area for these techniques is termination analysis of functional programs. Since the evaluation mechanism of functional languages is mainly term rewriting, one can transform functional programs into TRSs and prove termination of the resulting TRSs to conclude termination of the functional programs [6]. Although "full" rewriting does not impose any evaluation strategy, this approach is sound even if the underlying programming language has an evaluation strategy.

But in order to detect bugs in programs, it is at least as important to prove *non-termination* of programs or of the corresponding TRSs. Here, the evaluation strategy cannot be ignored, because a non-terminating TRS may still be terminating when considering the strategy. Thus, in order to disprove termination of programming languages with strategies, it is important to develop automated techniques to disprove termination of TRSs under strategies.

Only a few techniques for showing non-termination of TRSs have been introduced so far [4,7,9,10,12]. These techniques can be used to detect loops—a specific form of derivation which implies non-termination—and are successfully implemented in many tools (e.g., AProVE[5], Jambox [2], Matchbox [16], NTI [12], TORPA [17], T$_\text{T}$T$_2$ [8]).

If one wants to prove non-termination under strategies then up to now there are two different approaches. The first one is to directly analyze the loops whether they also imply non-termination under a given strategy $\mathcal{S}$. This approach was successfully applied for the innermost strategy in [15] where a decision procedure was given to determine whether a loop is an innermost loop.

* This author is supported by FWF (Austrian Science Fund) project P18763.

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 17–31, 2009.
© Springer-Verlag Berlin Heidelberg 2009

The second approach is to use a complete transformation $\tau_{\mathcal{S}}$ for strategy $\mathcal{S}$ such that $\mathcal{R}$ is terminating under $\mathcal{S}$ iff $\tau_{\mathcal{S}}(\mathcal{R})$ is (innermost) terminating. Then one first applies the transformation and then searches for a loop in $\tau_{\mathcal{S}}(\mathcal{R})$ afterwards. Here, the methods of [3] and [13,14] are applicable which can be used to disprove context-sensitive and outermost termination.

Although the second approach of using the transformations [3,13,14] seems to be a good solution to disprove context-sensitive and outermost termination, there are two main drawbacks. The first problem is a practical one. Often the loops of $\mathcal{R}$ are transformed into much longer loops in $\tau_{\mathcal{S}}(\mathcal{R})$ and hence, the search space for loops may become critical. And even more severe is the problem, that some loops of $\mathcal{R}$ are not even translated to loops in $\tau_{\mathcal{S}}(\mathcal{R})$ and hence, one even looses power if the search problem for loops is ignored.

Thus, there is still need to extend the first approach—to ensure or even decide that a given loop is a loop under strategies—to other strategies besides innermost. To this end, in this paper we first generalize the notion of a loop and an innermost loop to a loop under some arbitrary strategy. Then we develop two new decision procedures for context-sensitive loops and outermost loops.

The paper is structured as follows. In Sect. 2 we recapitulate the required notions of rewriting and generalize the notion of a loop for rewriting strategies. Moreover, we present a decision procedure for the question whether a given loop is a context-sensitive loop. Then in Sect. 3 we show how to formulate the same question for the outermost strategy as a set of matching problems. How these matching problems can be transformed to a simpler kind of problems—identity problems—is the content of Sect. 4. Afterwards, in Sect. 5 we provide a decision procedure for solvability of identity problems. All of our techniques have been implemented in the Tyrolean Termination Tool 2 (T$_2$) and the empirical results are presented in Sect. 6, before we conclude in Sect. 7.

A full version of this paper containing all proofs is available at `http://cl-informatik.uibk.ac.at/~griff/experiments/lus.php`. This website also contains details about our experiments.

2 Loops

We only regard finite signatures and TRSs and refer to [1] for the basics of rewriting. We use $\ell, r, s, t, u, \ldots$ for terms, $f, g, \ldots$ for function symbols, $x, y, \ldots$ for variables, σ, μ for substitutions, i, j, k, n, m, o for natural numbers, $p, q, \ldots$ for positions where ε is the root position, and $C, D, \ldots$ for contexts. Here, contexts are terms which contain exactly one hole $\Box$. For contexts, the term $C[t]$ is like C where $\Box$ is replaced by t, i.e., $\Box[t] = t$ and $f(s_1, \ldots, C, \ldots, s_n)[t] = f(s_1, \ldots, C[t], \ldots, s_n)$. We write $t|_p$ for the subterm of t at position p, i.e., $t|_\varepsilon = t$ and $f(s_1, \ldots, s_n)|_{ip} = s_i|_p$. The set of variables is denoted by $\mathcal{V}$.

Throughout this paper we assume a fixed TRS $\mathcal{R}$ and we write $t \to_p s$ if one can reduce t to s at position p with $\mathcal{R}$, i.e., $t = C[\ell\sigma]$ and $s = C[r\sigma]$ for some $\ell \to r \in \mathcal{R}$, substitution σ, and context C with $C|_p = \Box$. Here, the term $\ell\sigma$ is called a redex at position p. The reduction is an outermost reduction, written

$t \xrightarrow{\circ}_p s$, iff t contains no redex at a position q above p (written $q < p$). If the position is irrelevant we just write $\to$ or $\xrightarrow{\circ}$. The TRS $\mathcal{R}$ is non-terminating iff there is an infinite derivation $t_1 \to t_2 \to \dots$. It is outermost non-terminating iff there is such an infinite derivation using $\xrightarrow{\circ}$ instead of $\to$.

An obvious approach to disprove termination is to search for a loop, i.e., a derivation where the starting term t is reduced to a term containing an instance of t, i.e., $t \to^+ C[t\mu]$. The corresponding infinite derivation is

$$t \to^+ C[t\mu] \to^+ C[C[t\mu]\mu] \to^+ \dots \to^+ C[C[\dots C[t\mu] \dots \mu]\mu] \to^+ \dots \qquad (\star)$$

where the derivation $t \to^+ C[t\mu]$ is repeated over and over again. This infinite derivation $(\star)$ is obtained because $\to$ is closed under both substitutions and contexts, also known as stability and monotonicity.

However, in general it is not clear whether $(\star)$ also is an infinite derivation if one considers a specific evaluation strategy $\mathcal{S}$. If this is the case then and only then we speak of an $\mathcal{S}$-loop.

To formally define an $\mathcal{S}$-loop we first need to make the derivations within $(\star)$ precise. Therefore, we must represent terms like $C[C[\dots C[t\mu] \dots \mu]\mu]$ without using "…". Moreover, we must know the positions of the reductions since several strategies—like innermost, outermost, or context-sensitive—only allow reductions at certain positions. To this end, we define the notion of a context-substitution which combines insertion into a context with the application of a substitution.

Definition 1 (Context-substitutions). *A* context-substitution *is a pair (C, μ) consisting of a context C and a substitution μ. The n-fold application of (C, μ) to a term t, written $t(C, \mu)^n$ is defined as follows.*

- $t(C, \mu)^0 = t$
- $t(C, \mu)^{n+1} = C[t(C, \mu)^n \mu]$

From the definition it is obvious that in $t(C, \mu)^n$ the context C is added n-times above t and t is instantiated by μ^n. Note that also the added contexts are instantiated by μ. For the term $t(C, \mu)^3$ this is illustrated in Fig. 1.

The following lemma shows that context-substitutions have similar properties to both contexts and substitutions.

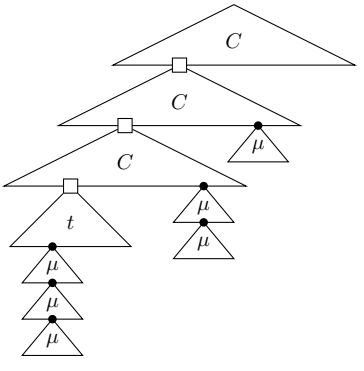


Fig. 1. The term $t(C, \mu)^3$

Lemma 1 (Properties of context-substitutions).

(i) $t(C, \mu)^n \mu = t\mu(C\mu, \mu)^n$.
(ii) $t(C, \mu)^m (C, \mu)^n = t(C, \mu)^{m+n}$.
(iii) *If* $C|_p = \Box$ *then* $t(C, \mu)^n|_{p^n} = t\mu^n$.
(iv) *Whenever* $t \to_q s$ *and* $C|_p = \Box$ *then* $t(C, \mu)^n \to_{p^n q} s(C, \mu)^n$.

Here, property (i) is similar to the fact that $C[t]\mu = C\mu[t\mu]$, and (ii) expresses that context-substitutions can be combined just as substitutions where $\sigma^m\sigma^n = \sigma^{m+n}$. Moreover, the property of contexts that $C[t]_p = t$ if $C|_p = \Box$ is extended in (iii), and finally stability and monotonicity of rewriting are used to show in (iv) that rewriting is closed under context-substitutions.

With the help of context-substitutions we can now describe the infinite derivation in ($\star$) more concisely. Since $t \to^+ C[t\mu] = t(C,\mu)$ we obtain

$$t(C,\mu)^0 \to^+ t(C,\mu)(C,\mu)^0 = t(C,\mu)^1 \to^+ \ldots \to^+ t(C,\mu)^n \to^+ \ldots \qquad (\star\star)$$

Hence, the terms that occur during the derivation are precisely defined and for every n the positions of the reductions are prefixed by an additional p^n where p is the position of the hole in C, cf. Lemma 1 (iv). In other words, every reduction takes place at the same position of the subterm $t\mu^n$ of $t(C,\mu)^n$.

Now it is natural to define that a derivation $t \to^+ t(C,\mu)$ is called an *$\mathcal{S}$-loop* iff all steps in ($\star\star$) respect the strategy $\mathcal{S}$.[1]

Definition 2 ($\mathcal{S}$-loops). *Let $\mathcal{S}$ be a strategy. A loop $t_1 \to_{q_1} t_2 \to_{q_2} \ldots t_n \to_{q_n} t_{n+1} = t_1(C,\mu)$ with $C|_p = \Box$ is an* $\mathcal{S}$-loop *iff all reductions $t_i(C,\mu)^m \to_{p^m q_i} t_{i+1}(C,\mu)^m$ respect the strategy $\mathcal{S}$ for all $1 \le i \le n$ and $m \ge 0$.*

As a direct consequence of Def. 2 one can conclude that every $\mathcal{S}$-loop of a rewrite system $\mathcal{R}$ proves non-termination of $\mathcal{R}$ under strategy $\mathcal{S}$.

Example 1. We consider the TRS $\mathcal{R}_n$ for arithmetic with n-bit numbers.

$$\mathsf{p}(\mathsf{0}) \to \mathsf{0} \quad (1) \qquad \mathsf{plus}(\mathsf{0}, y) \to y \quad (6)$$
$$\mathsf{p}(\mathsf{s}(x)) \to x \quad (2) \qquad \mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{plus}(x, y)) \quad (7)$$
$$\mathsf{minus}(x, \mathsf{0}) \to x \quad (3) \qquad \mathsf{inf} \to \mathsf{s}(\mathsf{inf}) \quad (8)$$
$$\mathsf{minus}(x, x) \to \mathsf{0} \quad (4) \qquad \mathsf{s}^{2^n}(x) \to \mathsf{overflow} \quad (9)$$
$$\mathsf{minus}(x, \mathsf{s}(y)) \to \mathsf{p}(\mathsf{minus}(x, y)) \quad (5)$$

Here, the last rule is used to model that an overflow occurred due to the n-bit restriction. We focus on the loops

$$t_1 = \mathsf{minus}(x, \mathsf{inf}) \to \mathsf{minus}(x, \mathsf{s}(\mathsf{inf})) \to \mathsf{p}(\mathsf{minus}(x, \mathsf{inf})) = C_1(t_1, \mu_1) \quad \text{and}$$
$$t_2 = \mathsf{plus}(\mathsf{inf}, y) \to \mathsf{plus}(\mathsf{s}(\mathsf{inf}), y) \to \mathsf{s}(\mathsf{plus}(\mathsf{inf}, y)) = C_2(t_2, \mu_2)$$

where $\mu_1 = \mu_2 = \{\}$, $C_1 = \mathsf{p}(\Box)$, and $C_2 = \mathsf{s}(\Box)$. Here, the first loop is an outermost loop, but the second one is not. The reason for the latter is that in every iteration one more s is created. Hence, this will lead to a redex w.r.t. Rule (9). Note that this example will be hard to handle with the transformational approaches: using [14] creates a TRS where all infinite reductions are non-looping and [13] is not even applicable due to non-left-linearity of Rule (4).

[1] Another natural definition of an $\mathcal{S}$-loop would just require that $t(C,\mu)^n \to^+ t(C,\mu)^{n+1}$ are $\mathcal{S}$-derivations for all n. This alternative was already used in the setting of dependency pairs in [4, Footnote 6]. However, there are problems using this definition which are described in [15, Sect. 2].

Note that a loop is not only determined by the precise derivation $t_1 \to^+ t_{n+1}$, but also by the specific context C that is used. To see this consider the TRS $\mathcal{R} = \{\mathsf{a} \to \mathsf{f}(\mathsf{a},\mathsf{a}), \mathsf{f}(\mathsf{f}(x,y),z) \to \mathsf{b}\}$. Then $\mathsf{a} \to \mathsf{f}(\mathsf{a},\mathsf{a})$ is a looping derivation. For $C = \mathsf{f}(\mathsf{a},\Box)$ this is an outermost loop. However, for $C' = \mathsf{f}(\Box,\mathsf{a})$ we do not obtain an outermost loop since $\mathsf{f}(\mathsf{a},\mathsf{a}) \xrightarrow{o} \mathsf{f}(\mathsf{f}(\mathsf{a},\mathsf{a}),\mathsf{a}) \not\xrightarrow{o} \mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a},\mathsf{a}),\mathsf{a}),\mathsf{a})$. Hence, the choice of C is essential.

For automatic non-termination analysis under strategies one main question is whether a given loop is an $\mathcal{S}$-loop, i.e., whether the loop implies non-termination even under strategy $\mathcal{S}$. In [15] it was already shown that this question is decidable for *innermost loops*.[2] There, one has the problem that innermost rewriting is not stable, although it is monotonic.

In context-sensitive rewriting [11] we have the inverse situation: first, stability is given whereas monotonicity is absent. And second, whereas the decision procedure for innermost loops is quite involved and already known, for *context-sensitive loops* we can present a novel decision procedure that is rather straightforward, but nevertheless important.

Theorem 1 (Deciding context-sensitive-loops). *A loop $t \to^+ C[t\mu]$ is a context-sensitive loop (using replacement map ν) iff both the derivation $t \to^+ C[t\mu]$ respects the context-sensitive strategy and the hole in C is at a ν-replacing position.*

Proof. Let $t_1 \to_{q_1} t_2 \to_{q_2} \dots t_n \to_{q_n} t_{n+1} = t_1(C,\mu)$ be a loop where $C|_p = \Box$. Then the following statements are all equivalent.

- the loop is a context-sensitive loop
- all $t_i(C,\mu)^m \to_{p^m q_i} t_{i+1}(C,\mu)^m$ are context-sensitive reductions
- all $p^m q_i$ are ν-replacing positions of $t_i(C,\mu)^m$
- p is a ν-replacing position of C and each q_i is a ν-replacing position of $t_i\mu^m$
- the hole in C is at a ν-replacing position and the derivation $t_1 \to^+ t_1(C,\mu)$ is a context-sensitive derivation ◻

In the rest of this paper we consider the outermost strategy. As main result we develop a decision procedure for the question whether a given loop is an *outermost loop*. Note that for outermost rewriting neither stability nor monotonicity are given. To see this consider the TRS $\mathcal{R} = \{\mathsf{a} \to \mathsf{a}, \mathsf{f}(x) \to x, \mathsf{g}(\mathsf{f}(\mathsf{a})) \to \mathsf{a}\}$. Then $\mathsf{a} \xrightarrow{o} \mathsf{a}$, but $\mathsf{f}(\mathsf{a}) \not\xrightarrow{o} \mathsf{f}(\mathsf{a})$. Moreover, $\mathsf{g}(\mathsf{f}(x)) \xrightarrow{o} \mathsf{g}(x)$, but $\mathsf{g}(\mathsf{f}(\mathsf{a})) \not\xrightarrow{o} \mathsf{g}(\mathsf{a})$.

The problem of missing stability was already present for innermost loops. Therefore, many techniques of [15] for innermost loops can be reused for outermost loops, too. However, to handle the missing monotonicity of outermost rewriting we have to extend these techniques by an additional context. And these contexts will require significant extensions of the techniques of [15] and are not so easy to treat as in the context-sensitive case.

[2] Note that in [15] one did not regard contexts, i.e., for an innermost loop one just required that all reductions $t_i\mu^m \to_{q_i} t_{i+1}\mu^m$ are innermost reductions. However, that definition of an innermost loop is equivalent to Def. 2 since innermost rewriting (denoted by $\xrightarrow{i}$) is monotonic. Thus, $t_i\mu^m \xrightarrow{i}_{q_i} t_{i+1}\mu^m$ iff $t_i(C,\mu)^m \xrightarrow{i}_{p^m q_i} t_{i+1}(C,\mu)^m$.

3 Deciding Outermost Loops

Recall the definition of an outermost reduction. An outermost reduction of t at position p requires that there is no redex at a position q above p, i.e., all subterms $t|_q$ with $q < p$ must not be *matched* by some left-hand side of a rule in $\mathcal{R}$. Hence, the question of an outermost reduction can be formulated as a question of matching.

However, we do not have to consider a single outermost reduction but we want to know whether each reduction of a term $t(C,\mu)^m$ at position $p^m q$ is an outermost reduction (where $C|_p = \Box$ and $q \in \mathcal{P}\mathsf{os}(t)$). Looking at Fig. 1 one sees that there are two different cases how to obtain a subterm at a position above $p^m q$ that is matched by some left-hand side. First, the subterm may be a subterm of $t\mu^m$. Or otherwise, the subterm starts within the context. For the former case we can reuse the so called *matching problems* [15, Def. 12] and for the latter we need an extended version of matching problems containing contexts.

Definition 3 ((Extended) matching problems). *A* matching problem *is a pair $(\mathcal{M},\mu)$ where $\mathcal{M}$ is a set of pairs of terms $s \gtrdot \ell$. It is* solvable *iff there is a* solution (k,σ) *such that for all $s \gtrdot \ell \in \mathcal{M}$ the equation $s\mu^k = \ell\sigma$ is satisfied.*

An extended matching problem *is a quintuple $(D \gtrdot \ell, C, t, \mathcal{M}, \mu)$. It is* solvable *iff there is a* solution (n,k,σ) *such that the equation $D[t(C,\mu)^n]\mu^k = \ell\sigma$ is satisfied and (k,σ) is a solution to the matching problem $(\mathcal{M},\mu)$.*

To simplify presentation we write $(D \gtrdot \ell, C, t, \mu)$ instead of $(D \gtrdot \ell, C, t, \varnothing, \mu)$ and we write $(s \gtrdot \ell, \mu)$ instead of $(\{s \gtrdot \ell\}, \mu)$. Moreover, we use the notion "matching problem" also for extended matching problems.

To check whether $t(C,\mu)^m$ has a redex above position $p^m q$ one can now construct a set of *initial matching problems*. Essentially, one considers matching problems for the subterms of t above q. Additionally, for each subterm of $t(C,\mu)^m$ that starts with a subcontext $C|_{p'}$ of C, we build an extended matching problem.

Definition 4 (Initial matching problems). *Let $t \to_q u$ be a reduction and (C,μ) be a context-substitution with $C|_p = \Box$. Then the following* initial matching problems *are created for this reduction and context-substitution.*

- $(t|_{p'} \gtrdot \ell, \mu)$ *for each $\ell \to r \in \mathcal{R}$ and $p' < q$*
- $(C|_{p'} \gtrdot \ell, C\mu, t\mu, \mu)$ *for each $\ell \to r \in \mathcal{R}$ and $p' < p$*

Example 2. Consider the loop $t_1 = \mathsf{minus}(x,\mathsf{inf}) \to_2 \mathsf{minus}(x,\mathsf{s}(\mathsf{inf})) = t_2 \to_\varepsilon \mathsf{p}(\mathsf{minus}(x,\mathsf{inf})) = t_1(C,\mu)$ of Ex. 1 where $C = \mathsf{p}(\Box)$ and $\mu = \{\}$. For the second reduction at root position we only build the extended matching problems $MP_{1\ell} = (\mathsf{p}(\Box) \gtrdot \ell, \mathsf{p}(\Box), \mathsf{minus}(x,\mathsf{s}(\mathsf{inf})), \mu)$ for all left-hand sides ℓ of $\mathcal{R}$. For the first reduction we obtain the similar extended matching problems $MP_{2\ell} = (\mathsf{p}(\Box) \gtrdot \ell, \mathsf{p}(\Box), \mathsf{minus}(x,\mathsf{inf}), \mu)$, but additionally we also get the matching problems $MP_{3\ell} = (\mathsf{minus}(x,\mathsf{inf}) \gtrdot \ell, \mu)$.

The following theorem states that we have setup the right initial matching problems. If we consider all initial problems of all reductions $t_i \to_{q_i} t_{i+1}$ of a loop,

then solvability of one of these problems is equivalent to the property that the loop is not outermost.

Theorem 2 (Outermost loops and matching problems). *Let $t \to_q u$ and (C, μ) be given such that $C|_p = \Box$. All reductions $t(C,\mu)^m \to_{p^m q} u(C,\mu)^m$ are outermost iff none of the initial matching problems for $t \to_q u$ and (C,μ) is solvable.*

Proof. We first prove that any solvable initial matching problem shows that there is at least one reduction of $t(C,\mu)^m$ at position $p^m q$ which is not an outermost reduction. There are two cases. First, if $(t|_{p'} \trianglerighteq \ell, \mu)$ is solvable, then there is a solution (k,σ) such that $t|_{p'}\mu^k = \ell\sigma$. Then $t(C,\mu)^k|_{p^k p'} = t\mu^k|_{p'} = t|_{p'}\mu^k = \ell\sigma$ shows that there is a redex in $t(C,\mu)^k$ above $p^k q$. Thus, $t(C,\mu)^k \to_{p^k q} u(C,\mu)^k$ is not an outermost reduction.

Otherwise, $(C|_{p'} \trianglerighteq \ell, C\mu, t\mu, \mu)$ is solvable. Hence, there is a solution (n,k,σ) such that $C|_{p'}[t\mu(C\mu,\mu)^n]\mu^k = \ell\sigma$. Here, we show that the term $t(C,\mu)^{n+1+k}$ has a redex at position $p^k p'$ which is above position $p^{n+1+k}q$:

$$\begin{aligned} t(C,\mu)^{n+1+k}|_{p^k p'} &= t(C,\mu)^n(C,\mu)(C,\mu)^k|_{p^k p'} &&= t(C,\mu)^n(C,\mu)\mu^k|_{p'} \\ &= C[t(C,\mu)^n\mu]\mu^k|_{p'} &&= C|_{p'}[t(C,\mu)^n\mu]\mu^k \\ &= C|_{p'}[t\mu(C\mu,\mu)^n]\mu^k &&= \ell\sigma \end{aligned}$$

For the other direction we show that if some reduction $t(C,\mu)^m$ at position $p^m q$ is not an outermost reduction, then one of the initial matching problems must be solvable. So suppose, the reduction of $t(C,\mu)^m$ is not outermost. Then there must be some position $q' < p^m q$ such that the corresponding subterm $t(C,\mu)^m|_{q'}$ is a redex $\ell\sigma$. Again, there are two cases.

First, if $q' \geq p^m$ then $q' = p^m p'$ where $p' < q$ as $q' < p^m q$. Hence, $\ell\sigma = t(C,\mu)^m|_{q'} = t(C,\mu)^m|_{p^m p'} = t\mu^m|_{p'} = t|_{p'}\mu^m$. Thus, the initial matching problem $(t|_{p'} \trianglerighteq \ell, \mu)$ has the solution (m,σ).

In the other case $q' < p^m$. Thus, we can split the position q' into $p^k p'$ where $k < m$ and $p' < p$. Moreover, there must be some $n \in \mathbb{N}$ that satisfies $m = n + 1 + k$. We conclude

$$\begin{aligned} \ell\sigma &= t(C,\mu)^m|_{q'} &&= t(C,\mu)^{n+1+k}|_{p^k p'} \\ &= t(C,\mu)^n(C,\mu)(C,\mu)^k|_{p^k p'} &&= t(C,\mu)^n(C,\mu)\mu^k|_{p'} \\ &= C[t(C,\mu)^n\mu]\mu^k|_{p'} &&= C[t\mu(C\mu,\mu)^n]\mu^k|_{p'} \\ &= C|_{p'}[t\mu(C\mu,\mu)^n]\mu^k. \end{aligned}$$

Thus, the initial matching problem $(C|_{p'} \trianglerighteq \ell, C\mu, t\mu, \mu)$ is solvable. □

Note that whenever $C = \Box$ then there is no initial matching problem which is an extended matching problem. Hence, by Thm. 2 one can already decide whether a loop $t \to^+ t\mu$ is an outermost loop by using the techniques of [15] to decide solvability of matching problems. For example it can be detected that all matching problems $MP_{3\ell}$ of Ex. 2 are not solvable.

However, in the general case we also generate extended matching problems. Therefore, in the next section we develop a novel decision procedure for solvability of extended matching problems like $MP_{1\ell}$ and $MP_{2\ell}$ of Ex. 2.

4 Deciding Solvability of Extended Matching Problems

Since extended matching problems are only generated if $C \neq \Box$, in the following sections we always assume that $C \neq \Box$. We take a similar approach to [15] where we transform each matching problem into $\top$, $\bot$, or into *solved form*. Here $\top$ and $\bot$ represent solvability and non-solvability. And if a matching problem is in solved form then often solvability can immediately be decided. We explain all transformation rules in detail directly after the following definition.

Definition 5 (Transformation of extended matching problems). *Let* $MP = (D \gtrdot \ell_0, C, t, \mathcal{M}, \mu)$ *be an extended matching problem where* $\mathcal{M} = \{s_1 \gtrdot \ell_1, \ldots, s_m \gtrdot \ell_m\}$. *Then* MP *is in* solved form *iff each* ℓ_i *is a variable. Let* $\mathcal{V}_{incr} = \{x \in \mathcal{V} \mid \exists n : x\mu^n \notin \mathcal{V}\}$ *be the set of increasing variables.*

We define a relation $\Rightarrow$ *which simplifies extended matching problems that are not in solved form. So, let* $\ell_j = f(\ell'_1, \ldots, \ell'_{m'})$.

(i) $MP \Rightarrow (D_{i'} \gtrdot \ell'_{i'}, C, t, \mathcal{M} \cup \{t_i \gtrdot \ell'_i \mid 1 \leq i \leq m', i \neq i'\}, \mu)$ *if* $j = 0$ *and* $D = f(t_1, \ldots, D_{i'}, \ldots, t_{m'})$.
(ii) $MP \Rightarrow (D \gtrdot \ell_0, C, t, (\mathcal{M} \setminus \{s_j \gtrdot \ell_j\}) \cup \{t_i \gtrdot \ell'_i \mid 1 \leq i \leq m'\}, \mu)$ *if* $j > 0$ *and* $s_j = f(t_1, \ldots, t_{m'})$.
(iii) $MP \Rightarrow \bot$ *if* $j = 0$ *and* $D = g(\ldots)$ *where* $f \neq g$.
(iv) $MP \Rightarrow \bot$ *if* $j > 0$ *and* $s_j = g(\ldots)$ *where* $f \neq g$.
(v) $MP \Rightarrow \bot$ *if* $j > 0$ *and* $s_j \in \mathcal{V} \setminus \mathcal{V}_{incr}$.
(vi) $MP \Rightarrow (D\mu \gtrdot \ell_0, C\mu, t\mu, \{s_i\mu \gtrdot \ell_i \mid 1 \leq i \leq m\}, \mu)$ *if* $j > 0$ *and* $s_j \in \mathcal{V}_{incr}$.
(vii) $MP \Rightarrow \top$ *if* $j = 0$, $D = \Box$, *and* $(\mathcal{M} \cup \{t \gtrdot \ell_0\}, \mu)$ *is solvable.*
(viii) $MP \Rightarrow (C \gtrdot \ell_0, C\mu, t\mu, \mathcal{M}, \mu)$ *if* $j = 0$, $D = \Box$, *and* $(\mathcal{M} \cup \{t \gtrdot \ell_0\}, \mu)$ *is not solvable.*

Recall that $MP = (D \gtrdot \ell_0, C, t, \{s_1 \gtrdot \ell_1, \ldots, s_m \gtrdot \ell_m\}, \mu)$ is solvable iff there is a solution (n, k, σ) such that $D[t(C, \mu)^n]\mu^k = \ell_0\sigma$ and $s_i\mu^k = \ell_i\sigma$ for all $1 \leq i \leq m$. Hence, whenever $D \neq \Box$ or $s_i \notin \mathcal{V}$ then one can perform a decomposition (Rules (i) and (ii)) or detect a clash (Rules (iii) and (iv)) as in a standard matching algorithm.

If $s_j = x$ is a non-increasing variable then $s_j\mu^k$ will always be a variable. Thus, Rule (v) correctly returns $\bot$. But if $s_j = x$ is an increasing variable then there might be a solution if $k > 0$. Hence, one can just apply μ once on the whole matching problem using Rule (vi). Note that in the result of Rule (vi) both t and C are also instantiated. This reflects the property of context-substitutions that $t(C, \mu)^n\mu = t\mu(C\mu, \mu)^n$, cf. Lemma 1.

Whereas Rule (v) and a simplified version of Rule (vi) have already been present in [15], here we also need two additional rules to handle contexts. Note that for $n = 0$ and $D = \Box$ the term $D[t(C, \mu)^n]\mu^k$ is just $t\mu^k$ and thus, one only has to consider a non-extended matching problem. Now, in Rules (vii) and (viii) there is a case distinction whether this non-extended matching problem is solvable, i.e., whether $n = 0$ yields a solution or not. If it is solvable then also a solution of MP is found and Rule (vii) correctly returns $\top$. If it is not possible

then there is only one way to continue: apply the context-substitution at least once, and this is exactly what Rule (viii) does.

Before we formally state the soundness of the transformation rules in Thm. 3 we illustrate their application on the extended matching problems of Ex. 2.

Example 3. We first consider $MP_{1\ell} = (\mathsf{p}(\Box) \gtrdot \ell, \mathsf{p}(\Box), \mathsf{minus}(x, \mathsf{s}(\mathsf{inf})), \mu)$. If ℓ is not one of the left-hand sides $\mathsf{p}(\mathsf{0})$ or $\mathsf{p}(\mathsf{s}(x))$ then $\bot$ is obtained by Rule (iii). If one considers $\ell = \mathsf{p}(\mathsf{0})$ then $(\mathsf{p}(\Box) \gtrdot \mathsf{p}(\mathsf{0}), \mathsf{p}(\Box), \mathsf{minus}(x, \mathsf{s}(\mathsf{inf})), \mu) \Rightarrow (\Box \gtrdot \mathsf{0}, \mathsf{p}(\Box), \mathsf{minus}(x, \mathsf{s}(\mathsf{inf})), \mu)$ by Rule (i). And as $(\mathsf{minus}(x, \mathsf{s}(\mathsf{inf})) \gtrdot \mathsf{0}, \mu)$ is not solvable, Rule (viii) yields $(\mathsf{p}(\Box) \gtrdot \mathsf{0}, \mathsf{p}(\Box), \mathsf{minus}(x, \mathsf{s}(\mathsf{inf})), \mu)$. Finally, an application of Rule (iii) returns $\bot$ and thereby shows that the matching problem is not solvable. Since the transformation for $\ell = \mathsf{p}(\mathsf{s}(x))$ also results in $\bot$, we have detected that none of the matching problems $MP_{1\ell}$ is solvable.

A similar transformation shows that none of the matching problems $MP_{2\ell}$ is solvable. Hence, the loop of Ex. 2 is an outermost loop.

Theorem 3 (Soundness and termination of the transformation rules).

(i) If $MP \Rightarrow \bot$ then MP is not solvable.
(ii) If $MP \Rightarrow \top$ then MP is solvable.
(iii) If $MP \Rightarrow MP'$ then MP is solvable iff MP' is solvable.
(iv) The relation $\Rightarrow$ is terminating and confluent.[3]

Using the above theorem allows us to transform any initial matching problem into $\bot$, $\top$, or into a matching problem in solved form. In the first two cases solvability is decided, but in the last case we still need a way to extract solvability. Note that these resulting matching problems are all of the form $MP = (D \gtrdot x_0, C, t, \{s_1 \gtrdot x_1, \ldots, s_m \gtrdot x_m\}, \mu)$ where each $x_i \in \mathcal{V}$. Note that if all x_i are different—which is always the case if one considers left-linear TRSs—then MP is trivially solvable. One just can choose the solution (n, k, σ) where $n = k = 0$ and $\sigma = \{x_0/D[t], x_1/s_1, \ldots, x_m/s_m\}$.

The only problem arises if for some $i \neq j$ we have $x_i = x_j$. Then to choose $\sigma(x_i) = \sigma(x_j)$ one has to know that $s_i\mu^k = s_j\mu^k$ for some k. This so called *identity problem* already occurred in [15]. However, if $i = 0$ then we have to answer a more difficult question, namely whether $D[t(C, \mu)^n]\mu^k = s_j\mu^k$. This new kind of problem is introduced as *extended identity problem*.

Definition 6 ((Extended) identity problems). *An* identity problem *is a pair $(s \approx s', \mu)$. It is* solvable *iff there is some k such that $s\mu^k = s'\mu^k$.*

An extended identity problem *is a quadruple $(D \approx s, \mu, C, t)$. It is* solvable *iff there is a* solution (n, k) *such that $D[t(C, \mu)^n]\mu^k = s\mu^k$.*

We now can transform matching problems in solved form into an equivalent set of (extended) identity problems.

Theorem 4 (Transforming matching problems into identity problems). *Let $MP = (D \gtrdot x, C, t, \{s_1 \gtrdot x_1, \ldots, s_m \gtrdot x_m\}, \mu)$ be a matching problem in solved form. It is solvable iff each of the following identity problems is solvable.*

[3] Here we need the assumption $C \neq \Box$. Otherwise, Rule (viii) would not terminate.

- $(D \approx s_i, \mu, C, t)$ *where* i *is the least index such that* $x = x_i$.
- $(s_i \approx s_j, \mu)$ *for all* j *where* $i < j$ *is the least index such that* $x_i = x_j$.

One might wonder why this theorem is sound as each solution of $(s_i \approx s_j, \mu)$ might yield a different k_{ij}. The key point is that the maximum of all these k_{ij}'s is a solution for all identity problems $(s_i \approx s_j, \mu)$.

Note that [15] describes a decision procedure for solvability of identity problems $(s \approx s', \mu)$. Hence, we can already decide solvability of matching problems $(D \gtrdot x, C, t, \mathcal{M}, \mu)$ in solved form where the variable x does not occur in $\mathcal{M}$. Nevertheless, for the general case we still need a technique to decide solvability of extended identity problems. Such a technique is described in the next section.

Example 4. Consider the TRS $\{\mathsf{f}(x) \to \mathsf{g}(\mathsf{g}(x,x), \mathsf{f}(\mathsf{s}(x))), \mathsf{g}(y,y) \to \mathsf{a}\}$ with the loop $t = \mathsf{f}(x) \to \mathsf{g}(\mathsf{g}(x,x), \mathsf{f}(\mathsf{s}(x))) = t(C, \mu)$ where $\mu = \{x/\mathsf{s}(x)\}$ and $C = \mathsf{g}(\mathsf{g}(x,x), \Box)$. One initial matching problem $(\mathsf{g}(\mathsf{g}(x,x), \Box) \gtrdot \mathsf{f}(x), C\mu, t\mu, \mu)$ is trivially not solvable due to a symbol clash. But the other initial matching problem $(\mathsf{g}(\mathsf{g}(x,x), \Box) \gtrdot \mathsf{g}(y,y), C\mu, t\mu, \mu)$ is transformed into the matching problem $MP = (\Box \gtrdot y, C\mu, t\mu, \{\mathsf{g}(x,x) \gtrdot y\}, \mu)$. By Thm. 4 solvability of MP is equivalent to solvability of the extended identity problem $(\Box \approx \mathsf{g}(x,x), \mu, C\mu, t\mu)$.

5 Deciding Solvability of Extended Identity Problems

In this section we describe a decision procedure for solvability of extended identity problems. To this end we first introduce the notion of a trace.

Definition 7 (Traces). *The* trace *of term* t *w.r.t. position* p *is the sequence of function symbols and indices that are passed when moving from* ε *to* p *in* t*:*

$$\mathsf{trace}(p,t) = \begin{cases} \varepsilon & \text{if } t = x \text{ or } p = \varepsilon \\ f\, i\, \mathsf{trace}(q, t_i) & \text{if } p = iq \text{ and } t = f(t_1, \ldots, t_n) \end{cases}$$

The trace of a context C *with* $C|_p = \Box$ *is* $\mathsf{trace}(C) = \mathsf{trace}(p, C)$. *The set of all* traces of a term *is* $\mathsf{Traces}(t) = \{\mathsf{trace}(p,t) \mid p \in \mathcal{P}\mathsf{os}(t)\}$.

Lemma 2 (Properties of traces).

(i) $\mathsf{trace}(pq, C[t]) = \mathsf{trace}(C)\mathsf{trace}(q,t)$ *if* $C|_p = \Box$
(ii) $\mathsf{trace}(p,t) = \mathsf{trace}(p, t\mu)$ *if* $p \in \mathcal{P}\mathsf{os}(t)$
(iii) $\mathsf{trace}(p^n q, t(C,\mu)^n) = \mathsf{trace}(C)^n \mathsf{trace}(q, t\mu^n)$ *if* $C|_p = \Box$ *and* $q \in \mathcal{P}\mathsf{os}(t)$
(iv) $\mathsf{trace}(p,t) \in \mathsf{Traces}(t)$ *if* $\mathsf{trace}(pq,t) \in \mathsf{Traces}(t)$

In the following algorithm to decide solvability of extended identity problems, a Boolean disjunction over non-extended identity problems represents solvability of at least one of these identity problems.

Definition 8 (Decision procedure for extended identity problems). *Let* $(D \approx s, \mu, C, t)$ *be an extended identity problem where* $D|_q = \Box$, $C|_p = \Box$.

(i) *if* $\mathsf{trace}(D)\mathsf{trace}(C)^* \not\subseteq \bigcup_{i\in\mathbb{N}} \mathsf{Traces}(s\mu^i) =: S$ *then there is some* m *such that* $\mathsf{trace}(D)\mathsf{trace}(C)^m \notin S$*; return* $\bigvee_{n<m} (D[t(C,\mu)^n] \approx s, \mu)$

(ii) *if* $\mathsf{trace}(C)^* \not\subseteq \bigcup_{i\in\mathbb{N}} \mathsf{Traces}(t\mu^i)$ *then return "not solvable"*

(iii) *let* x *be a variable which infinitely often occurs in* $s|_{p_0}, s\mu^1|_{p_1}, s\mu^2|_{p_2}, \ldots$ *where each* p_i *is that prefix of* qp^ω *which satisfies* $p_i \in \mathcal{P}\mathsf{os}(s\mu^i)$*; let* i *be the minimal number such that* $s\mu^i|_{p_i} = x$

(iv) *let* j *be minimal such that* $t\mu^j|_{q_j} = x$ *where* q_j *is that prefix of* p^ω *which satisfies* $q_j \in \mathcal{P}\mathsf{os}(t\mu^j)$*; if there is no such* j *then return "not solvable"*

(v) *return* $\bigvee_{n\leq \max(\frac{|p_i|-|qq_j|}{|p|},j)} (D[t(C,\mu)^n] \approx s, \mu)$

We will explain the algorithm in detail within the proof of the following theorem. Afterwards, we present algorithms to automate the non-trivial steps.

Theorem 5. *The algorithm of Def. 8 is sound and terminates.*

Proof. Termination of the algorithm is obvious. We only remark that the disjunction in Step (v) is finite, since $|p| > 0$ by the assumption $C \neq \Box$.

To show soundness of the algorithm first recall the definition of solvability of $(D \approx s, \mu, C, t)$. This extended identity problem is solvable iff there is a solution (n, k) such that $D[t(C,\mu)^n]\mu^k = s\mu^k$. We observe two properties: first, whenever (n, k) is a solution then $(n, k + k')$ is also a solution. And second, if we fix n then the extended identity problem is solvable iff the identity problem $(D[t(C,\mu)^n] \approx s, \mu)$ is solvable. From the second observation we conclude that if one can bound the value of n, then one can reduce solvability of extended identity problems to solvability of identity problems and is done. And computing these bounds on n is basically all the algorithm does (in Steps (i) and (v)).

The first idea to extract a bound is to consider how the term $D[t(C,\mu)^n]\mu^k$ grows if n is increased. Looking at Fig. 1 on page 19 or using Lemma 2 we see that $D[t(C,\mu)^n]\mu^k$ has the trace $\mathsf{trace}(D)\mathsf{trace}(C)^n$. Thus, if (n, k) is a solution then $s\mu^k$ must have the same trace. Hence, if $\mathsf{trace}(D)\mathsf{trace}(C)^m \notin \bigcup_{i\in\mathbb{N}} \mathsf{Traces}(s\mu^i) = S$ then $n < m$. This proves soundness of Step (i).

So, after Step (i) we can assume $\mathsf{trace}(D)\mathsf{trace}(C)^* \subseteq S$. Hence, if we increase the k of $s\mu^k$ then this term grows along the (infinite) trace $\mathsf{trace}(D)\mathsf{trace}(C)^\omega$. Using the first observation we know that for every solution of the extended identity problem we can increase k arbitrarily. Thus, $D[t(C,\mu)^n]\mu^k$ (which is the same term as $s\mu^k$) also has to contain longer and longer parts of the trace $\mathsf{trace}(D)\mathsf{trace}(C)^\omega$ when increasing k. Hence, whenever the extended identity problem is solvable then $\mathsf{trace}(C)^* \subseteq \bigcup_{i\in\mathbb{N}} \mathsf{Traces}(t\mu^i) =: T$ which shows soundness of Step (ii).

If the decision procedure arrives at Step (iii) then both $\mathsf{trace}(D)\mathsf{trace}(C)^* \subseteq S$ and $\mathsf{trace}(C)^* \subseteq T$, i.e., when increasing k we see no difference of function symbols of the terms $D[t(C,\mu)^n]\mu^k$ and $s\mu^k$ along the path qp^ω. However, there still might be a difference between $D[t(C,\mu)^n]\mu^k$ and $s\mu^k$ for each finite value k. For example, different variables may be used to increase the terms along the path qp^ω as in $D = \Box, C = f(\Box), t = x, s = y, \mu = \{x/f(x), y/f(y)\}$. Or one of the terms is always a bit larger as the other one as in $D = \Box, C = f(\Box), t =$

$f(x), s = x, \mu = \{x/f(x)\}$. To detect the situation of different variables, Step (iv) is used, and the latter situation is done via Step (v).

As we are interested in the variables in Steps (iv) and (v), we first compute a variable x in Step (iii) which infinitely often occurs along the path qp^ω in the terms $s, s\mu, s\mu^2, \ldots$. This variable must exist, since μ has finite domain and $\mathsf{trace}(D)\mathsf{trace}(C)^* \subseteq S$. This immediately proves soundness of Step (iv) since whenever $D[t(C,\mu)^n]\mu^k = s\mu^k$ then by the first observation we can choose k high enough such that $x = s\mu^k|_{p_k} = D[t(C,\mu)^n]\mu^k|_{p_k}$ where $p_k \leq qp^\omega$, i.e., p_k must be of the form qp^nq' where q' is a prefix of p^ω. Thus, $x = D[t(C,\mu)^n]\mu^k|_{qp^nq'} = t(C,\mu)^n\mu^k|_{p^nq'} = t\mu^{n+k}|_{q'}$ shows that Step (iv) cannot stop the algorithm with "not solvable".

The main idea of Step (v) is as in Step (i) to bound n but now for a different reason. Observe that whenever we increase n then $s\mu^k$ stays the same whereas $D[t(C,\mu)^n]\mu^k$ has the subterm $t\mu^{n+k}$ which depends on n. And since $\mathsf{trace}(C)^* \subseteq T$ we know that with larger n also the terms $t\mu^{n+k}$ become larger. Thus, there must be a limit where the size of $s\mu^k$ is reached and it is of no use to search for larger values of n. And this limit turns out to be $m := \max(\frac{|p_i|-|qq_j|}{|p|}, j)$ which proves soundness of Step (v). □

Input: D, C, s, μ
Output: minimal m such that $\mathsf{trace}(D)\mathsf{trace}(C)^m \notin \bigcup_{i\in\mathbb{N}} \mathsf{Traces}(s\mu^i)$ or ∞, otherwise

(1) $m := 0$, $E := D$, $t := s$, $S := \varnothing$
(2) if $E = f(\ldots E' \ldots)$ and $t = f(\ldots)$ where $E|_i = E'$ then $E := E'$, $t := t|_i$, goto (2)
(3) if $E = g(\ldots)$, $t = f(\ldots)$, and $g \neq f$ then return m
(4) if $E \neq \Box$ and $t = x \notin \mathcal{V}_{\mathsf{incr}}(\mu)$ then return m
(5) if $E \neq \Box$ and $t = x \in \mathcal{V}_{\mathsf{incr}}(\mu)$ then $t := t\mu$, goto (2)
(6) if $E = \Box$ and $t \in S$ then return ∞
(7) if $E = \Box$ and $t \notin S$ then $S := S \cup \{t\}$, $m := m + 1$, $E := C$, goto (2)

Fig. 2. $\mathsf{clash}(D, C, s, \mu)$

For the automation one can use the algorithm of [15] for solvability of non-extended identity problems. To check whether $\mathsf{trace}(D)\mathsf{trace}(C)^* \subseteq S$ in Step (i) one can check whether $\mathsf{clash}(D, C, s, \mu) = \infty$ (cf. Fig. 2) which also delivers the required number m in case that $\mathsf{trace}(D)\mathsf{trace}(C)^* \not\subseteq S$. Of course, clash can also be used for the test in Step (ii) where we call $\mathsf{clash}(\Box, C, t, \mu)$.

Theorem 6. *The algorithm* clash *terminates and is sound.*

For Step (iii) of the decision procedure the function $\mathsf{var}_\infty(s, q, p, \mu)$ in Fig. 3 computes a triple (x, p_i, i) such that x infinitely often occurs in the sequence $s|_{p_0}$, $s\mu|_{p_1}$, $s\mu^2|_{p_2}$, ... where each $p_k \in \mathcal{P}\mathsf{os}(s\mu^k)$ is the maximal prefix of qp^∞ and i is the smallest number such that $s\mu^i|_{p_i} = x$. Note that the precondition is satisfied, since var_∞ is only called if $\mathsf{trace}(D)\mathsf{trace}(C)^* \subseteq \bigcup_{i\in\mathbb{N}} \mathsf{Traces}(s\mu^i)$ which directly implies $qp^* \subseteq \bigcup_{i\in\mathbb{N}} \mathcal{P}\mathsf{os}(s\mu^i)$.

Preconditions: $p \neq \varepsilon$ and $qp^* \subseteq \bigcup_{j\in\mathbb{N}} \mathcal{P}\mathsf{os}(s\mu^j)$
Input: s, q, p, μ
Output: (x, p_i, i) such that x infinitely often occurs in terms $s\mu^j$ along path qp^ω where i and $p_i < qp^\omega$ are minimal such that $s\mu^i|_{p_i} = x$

(1) $i := 0$, $u := s$, $p_{\mathsf{start}} := \varepsilon$, $p_{\mathsf{end}} := q$, $S := \varnothing$
(2) if $u = x$ and $(x, _, p_{\mathsf{end}}, _) \in S$ then return $(x, p_{\mathsf{start}}{}', i')$ where i' is the smallest value such that $(x, p_{\mathsf{start}}{}', _, i') \in S$
(3) if $u = x$ and $(x, _, p_{\mathsf{end}}, _) \notin S$ then $u := u\mu$, $S := S \cup \{(x, p_{\mathsf{start}}, p_{\mathsf{end}}, i)\}$, $i := i + 1$, goto (2)
(4) (a) if $p_{\mathsf{end}} = \varepsilon$ then $p_{\mathsf{end}} := p$
(b) let $p_{\mathsf{end}} = jp'$ and $u = f(u_1, \ldots, u_n)$; $u := u_j$, $p_{\mathsf{end}} := p'$, $p_{\mathsf{start}} := p_{\mathsf{start}}j$, goto (2)

Fig. 3. $\mathsf{var}_\infty(s, q, p, \mu)$

Theorem 7. *The algorithm* var_∞ *terminates and is sound.*

Preconditions: $p \neq \varepsilon$ and $p^* \subseteq \bigcup_{i\in\mathbb{N}} \mathcal{P}\mathsf{os}(t\mu^i)$
Input: x, t, p, μ
Output: (j, q) if j is minimal such that $t\mu^j|_q = x$ where $q \in \mathcal{P}\mathsf{os}(t\mu^j)$ is prefix of p^ω or $\bot$, if there is no such j

(1) $j := 0$, $u := t$, $p_{\mathsf{start}} := \varepsilon$, $p_{\mathsf{end}} := \varepsilon$, $S := \varnothing$
(2) if $u = x$ then return (j, p_{start})
(3) if $u = y \neq x$ and $(y, p_{\mathsf{end}}) \in S$ then return $\bot$
(4) if $u = y \neq x$ and $(y, p_{\mathsf{end}}) \notin S$ then $j := j+1$, $S := S \cup \{(y, p_{\mathsf{end}})\}$, $u := u\mu$, goto (2)
(5) (a) if $p_{\mathsf{end}} = \varepsilon$ then $p_{\mathsf{end}} := p$
(b) let $p_{\mathsf{end}} = ip'$ and $u = f(u_1, \ldots, u_n)$; $u := u_i$, $p_{\mathsf{end}} := p'$, $p_{\mathsf{start}} := p_{\mathsf{start}}i$, goto (2)

Fig. 4. $\mathsf{idx}(x, t, p, \mu)$

Finally, for Step (iv) a small adaptation of var_∞ yields the last required algorithm idx in Fig. 4 to check whether x occurs along p^ω in some term $t\mu^j$.

Theorem 8. *The algorithm* idx *terminates and is sound.*

Note that both algorithms var_∞ and idx become unsound if one only considers equal variables in the set S, but not equal positions p_{end}.

6 Empirical Results

In the following we first give some details about the actual implementation of our method and after that empirical results. To find loops, we use unfoldings as defined in [12], Section 3 (without any refinements mentioned in later sections).

For efficiency reasons we restrict to non-variable positions. Further we do not use a combination of forward and backward unfoldings by default. Our basic method uses the following heuristic to decide the direction of the unfoldings: For systems that are duplicating but whose inverse is non-duplicating we unfold backwards. For all other systems we unfold forwards.

In contrast to finding loops for full termination, for a specific strategy $\mathcal{S}$, we cannot always stop when a loop was found (since it could turn out to be no longer relevant when switching from full rewriting to $\mathcal{S}$-rewriting). Hence we compute a lazy list of potential loops that is checked one at a time corresponding to $\mathcal{S}$. If the checked loop is no $\mathcal{S}$-loop, the next loop is requested (and lazyness makes sure that the necessary computations are only done after the previous element dropped out).

For context-sensitive rewriting as well as outermost rewriting we reduce the search space by filtering the set of unfoldings after each iteration. In both cases we remove derivations containing rewrite steps that disobey the strategy.

As already mentioned in the introduction there is the transformational approach for both, context-sensitive rewriting [3] and outermost rewriting [13,14]. In both cases a problem for finding loops, is that the length of an existing loop may increase dramatically, or even worse, a loop is transformed into a non-looping infinite derivation.

To evaluate our implementation in T$_\mathsf{T}$T$_2$ we used all 291 (214) outermost examples as well as all 109 (15) context-sensitive examples of version 5.0.2 of the Termination Problems Data Base.[4] Here, in brackets the number of those TRSs is given, where outermost resp. context-sensitive termination has not already been proven. The results on these possibly non-terminating TRSs are as follows:

	outermost TRSs			CSRs
	AProVE	TrafO	T$_\mathsf{T}$T$_2$	T$_\mathsf{T}$T$_2$
NO score	37	30	191	4
avg. time (msec)	6689	6772	340	38

For outermost rewriting we compare to the sum of non-termination proofs (NO score) achieved by AProVE and TrafO[5] at the January 2009 termination competition.[46] The success of our technique is clearly visible: T$_\mathsf{T}$T$_2$ was able to disprove termination of nearly 90 % of all possible non-terminating TRSs, including all examples that could be handled by AProVE and TrafO (which use the transformational approaches of [14] and [13] respectively).

For context-sensitive rewriting we just give the NOs of T$_\mathsf{T}$T$_2$ since we are not aware of any other tool that has disproven context-sensitive termination of a single TRS. Here, our implementation could at least solve one quarter of the potentially non-terminating TRSs.

[4] `http://termcomp.uibk.ac.at`

[5] `http://www.win.tue.nl/~mraffels/trafo.html`

[6] The numbers for T$_\mathsf{T}$T$_2$ differ from those of the competition since the competition version did not feature the reduction of the search space which is described above.

7 Conclusion and Future Work

To prove non-termination of rewriting under strategy $\mathcal{S}$, we first extended the notion of a loop to an $\mathcal{S}$-loop. An $\mathcal{S}$-loop is an $\mathcal{S}$-reduction with a strong regularity which admits the same infinite reduction as an ordinary loop does for full rewriting. Afterwards, we developed two novel procedures to decide whether a given loop is a context-sensitive loop or an outermost loop. It is easy to see that the conjunction of both procedures decides context-sensitive outermost loops.

Since [6] only describes a way to prove termination of Haskell programs, it might be an interesting future work to combine our technique for outermost loops with [6] to also disprove termination of Haskell programs.

References

1. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
2. Endrullis, J.: Jambox, `http://joerg.endrullis.de`
3. Giesl, J., Middeldorp, A.: Transformation techniques for context-sensitive rewrite systems. Journal of Functional Programming 14(4), 379–427 (2004)
4. Giesl, J., Thiemann, R., Schneider-Kamp, P.: Proving and disproving termination of higher-order functions. In: Gramlich, B. (ed.) FroCos 2005. LNCS (LNAI), vol. 3717, pp. 216–231. Springer, Heidelberg (2005)
5. Giesl, J., Schneider-Kamp, P., Thiemann, R.: AProVE 1.2: Automatic termination proofs in the DP framework. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS, vol. 4130, pp. 281–286. Springer, Heidelberg (2006)
6. Giesl, J., Swiderski, S., Schneider-Kamp, P., Thiemann, R.: Automated termination analysis for Haskell: From term rewriting to programming languages. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 297–312. Springer, Heidelberg (2006)
7. Guttag, J., Kapur, D., Musser, D.: On proving uniform termination and restricted termination of rewriting systems. SIAM J. Computation 12, 189–214 (1983)
8. Korp, M., Sternagel, C., Zankl, H., Middeldorp, A.: Tyrolean Termination Tool 2. In: Treinen, R. (ed.) RTA 2009. LNCS, vol. 5595, pp. 295–304. Springer, Heidelberg (2009)
9. Kurth, W.: Termination und Konfluenz von Semi-Thue-Systemen mit nur einer Regel. PhD thesis, Technische Universität Clausthal, Germany (1990)
10. Lankford, D., Musser, D.: A finite termination criterion. Unpublished Draft. USC Information Sciences Institute (1978)
11. Lucas, S.: Context-sensitive computations in functional and functional logic programs. Journal of Functional and Logic Programming 1, 1–61 (1998)
12. Payet, É.: Loop detection in term rewriting using the eliminating unfoldings. Theoretical Computer Science 403(2-3), 307–327 (2008)
13. Raffelsieper, M., Zantema, H.: A transformational approach to prove outermost termination automatically. In: Proc. WRS 2008. ENTCS 237, pp. 3–21 (2009)
14. Thiemann, R.: From outermost termination to innermost termination. In: Nielsen, M., et al. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 533–545. Springer, Heidelberg (2009)
15. Thiemann, R., Giesl, J., Schneider-Kamp, P.: Deciding innermost loops. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 366–380. Springer, Heidelberg (2008)
16. Waldmann, J.: Matchbox: A tool for match-bounded string rewriting. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 85–94. Springer, Heidelberg (2004)
17. Zantema, H.: Termination of string rewriting proved automatically. Journal of Automated Reasoning 34, 105–139 (2005)

Proving Termination of Integer Term Rewriting*

Carsten Fuhs[1], Jürgen Giesl[1], Martin Plücker[1], Peter Schneider-Kamp[2], and Stephan Falke[3]

[1] LuFG Informatik 2, RWTH Aachen University, Germany
[2] Dept. of Mathematics & CS, University of Southern Denmark, Odense, Denmark
[3] CS Department, University of New Mexico, Albuquerque, NM, USA

Abstract. When using rewrite techniques for termination analysis of programs, a main problem are pre-defined data types like integers. We extend term rewriting by built-in integers and adapt the dependency pair framework to prove termination of *integer term rewriting* automatically.

1 Introduction

Recently, techniques and tools from term rewriting have been successfully applied to prove termination automatically for different programming languages, cf. e.g. [14,27]. The advantage of rewrite techniques is that they are very powerful for algorithms on user-defined data structures, since they can automatically generate suitable well-founded orders comparing arbitrary forms of terms. But in contrast to techniques for termination of imperative programs (e.g., [2,3,4,5,6,7,8,24,25]),[1] the drawback of rewrite techniques is that they do not support data structures like integer numbers which are pre-defined in almost all programming languages. Up to now, integers have to be represented as terms, e.g., using the symbols $\mathsf{0}$ for zero, s for the successor function, and pos and neg to convert natural to integer numbers. Then the integers 1 and -2 are represented by the terms $\mathsf{pos}(\mathsf{s}(\mathsf{0}))$ resp. $\mathsf{neg}(\mathsf{s}(\mathsf{s}(\mathsf{0})))$ and one has to add rules for pre-defined operations like $+$, $-$, $*$, $/$, $\%$ that operate on these terms. This representation leads to efficiency problems for large numbers and it makes termination proofs difficult. Therefore up to now, termination tools for term rewrite systems (TRSs) were not very powerful for algorithms on integers, cf. Sect. 6. Hence, an extension of TRS termination techniques to built-in data structures is one of the main challenges in the area [26].

To solve this challenge, we extend[2] TRSs by built-in integers in Sect. 2 and adapt the popular dependency pair (DP) framework for termination of TRSs to integers in Sect. 3. This combines the power of TRS techniques on user-defined data types with a powerful treatment of pre-defined integers. In Sect. 4, we improve the main *reduction pair processor* of the adapted DP framework by considering *conditions* and show how to simplify the resulting conditional constraints.

* Supported by the DFG grant GI 274/5-2 and by the G.I.F. grant 966-116.6.

[1] Moreover, integers were also studied in termination analysis for logic programs [28].

[2] First steps in this direction were done in [9], but [9] only integrated natural instead of integer numbers, which is substantially easier. Moreover, [9] imposed several restrictions (e.g., they did not integrate multiplication and division of numbers and disallowed conditions with mixtures of pre-defined and user-defined functions).

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 32–47, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Sect. 5 explains how to transform these conditional constraints into Diophantine constraints in order to generate suitable orders for termination proofs of integer TRSs (ITRSs). Sect. 6 evaluates our implementation in the prover AProVE [15].

2 Integer Term Rewriting

To handle integers in rewriting, we now represent each integer by a pre-defined constant of the same name. So the signature is split into two disjoint subsets $\mathcal{F}$ and $\mathcal{F}_{int}$. $\mathcal{F}_{int}$ contains the integers $\mathbb{Z} = \{0, 1, -1, 2, -2, \ldots\}$, the Boolean values $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$, and pre-defined operations. These operations are classified into *arithmetic operations* like $+$ which yield an integer when applied to integers, *relational operations* like $>$ which yield true or false when applied to integers, and *Boolean operations* like $\wedge$ which yield true or false when applied to Booleans.

Every ITRS implicitly contains an infinite set of pre-defined rules $\mathcal{PD}$ in order to evaluate the pre-defined operations on integers and Booleans. For example, the set $\mathcal{PD}$ contains the rules $2{*}21 \to 42$, $42 \geqslant 23 \to \mathsf{true}$, and $\mathsf{true} \wedge \mathsf{false} \to \mathsf{false}$.

These pre-defined operations can only be evaluated if both their arguments are integers resp. Booleans. So terms like $1 + x$ and $1 + \mathsf{true}$ are normal forms. Moreover, "$t/0$" and "$t \,\%\, 0$" are also normal forms for all terms t. As in most programming languages, an ITRS $\mathcal{R}$ may not have rules $\ell \to r$ where ℓ contains pre-defined operations or where $\ell \in \mathbb{Z} \cup \mathbb{B}$. The rewrite relation for an ITRS $\mathcal{R}$ is defined by simply considering innermost[3] rewriting with the TRS $\mathcal{R} \cup \mathcal{PD}$.

Definition 1 (ITRS). *Let* $\mathcal{A}rith\mathcal{O}p = \{+, -, *, /, \%\}$, $\mathcal{R}el\mathcal{O}p = \{>, \geqslant, <, \leqslant, ==, !=\}$, *and* $\mathcal{B}ool\mathcal{O}p = \{\wedge, \Rightarrow\}$.[4] *Moreover,* $\mathcal{F}_{int} = \mathbb{Z} \cup \mathbb{B} \cup \mathcal{A}rith\mathcal{O}p \cup \mathcal{R}el\mathcal{O}p \cup \mathcal{B}ool\mathcal{O}p$. *An* ITRS $\mathcal{R}$ *is a (finite) TRS over* $\mathcal{F} \uplus \mathcal{F}_{int}$ *where for all rules* $\ell \to r$, *we have* $\ell \in \mathcal{T}(\mathcal{F} \cup \mathbb{Z} \cup \mathbb{B}, \mathcal{V})$ *and* $\ell \notin \mathbb{Z} \cup \mathbb{B}$. *As usual,* $\mathcal{V}$ *contains all variables. The rewrite relation* $\hookrightarrow_{\mathcal{R}}$ *of an ITRS* $\mathcal{R}$ *is defined as* $\xrightarrow{\mathsf{i}}_{\mathcal{R} \cup \mathcal{PD}}$, *where*

$$\begin{aligned}\mathcal{PD} = \quad & \{n \circ m \to q \mid n, m, q \in \mathbb{Z},\ n \circ m = q,\ \circ \in \mathcal{A}rith\mathcal{O}p\} \\ \cup\ & \{n \circ m \to q \mid n, m \in \mathbb{Z},\ q \in \mathbb{B},\ n \circ m = q,\ \circ \in \mathcal{R}el\mathcal{O}p\} \\ \cup\ & \{n \circ m \to q \mid n, m, q \in \mathbb{B},\ n \circ m = q,\ \circ \in \mathcal{B}ool\mathcal{O}p\}\end{aligned}$$

[3] In this paper, we restrict ourselves to innermost rewriting for simplicity. This is not a severe restriction as innermost termination is equivalent to full termination for non-overlapping TRSs and moreover, many programming languages already have an innermost evaluation strategy. Even for lazy languages like Haskell, with the translation of programs to TRSs in [14], it suffices to show innermost termination.

[4] Of course, one could easily include additional junctors like $\vee$ or $\neg$ in $\mathcal{B}ool\mathcal{O}p$. Moreover, one could also admit ITRSs with conditions and indeed, our implementation also works on *conditional* ITRSs. This is no additional difficulty, because conditional (I)TRSs can be automatically transformed into unconditional ones [23]. E.g., the ITRS $\mathcal{R}_1$ below could result from the transformation of this conditional ITRS:

$$\begin{array}{ll}\mathsf{sum}(x, y) \to y + \mathsf{sum}(x, y + 1) & \mid\ x \geqslant y \to^* \mathsf{true} \\ \mathsf{sum}(x, y) \to 0 & \mid\ x \geqslant y \to^* \mathsf{false}\end{array}$$

For example, consider the ITRSs $\mathcal{R}_1 = \{(1), (2), (3)\}$ and $\mathcal{R}_2 = \{(4), (5), (6)\}$. Here, $\mathsf{sum}(x, y)$ computes $\sum_{i=y}^{x} i$ and $\mathsf{log}(x, y)$ computes $\lfloor log_y(x) \rfloor$.

$$\mathsf{sum}(x,y) \to \mathsf{sif}(x \geqslant y, x, y) \quad (1) \qquad \mathsf{log}(x,y) \to \mathsf{lif}(x \geqslant y \wedge y > 1, x, y) \quad (4)$$
$$\mathsf{sif}(\mathsf{true}, x, y) \to y + \mathsf{sum}(x, y+1) \quad (2) \qquad \mathsf{lif}(\mathsf{true}, x, y) \to 1 + \mathsf{log}(x/y, y) \quad (5)$$
$$\mathsf{sif}(\mathsf{false}, x, y) \to 0 \quad (3) \qquad \mathsf{lif}(\mathsf{false}, x, y) \to 0 \quad (6)$$

The term $\mathsf{sum}(1, 1)$ can be rewritten as follows (redexes are underlined):

$$\begin{aligned} \underline{\mathsf{sum}(1,1)} &\hookrightarrow_{\mathcal{R}_1} \mathsf{sif}(\underline{1 \geqslant 1}, 1, 1) \hookrightarrow_{\mathcal{R}_1} \underline{\mathsf{sif}(\mathsf{true}, 1, 1)} \hookrightarrow_{\mathcal{R}_1} 1 + \mathsf{sum}(1, \underline{1+1}) \\ &\hookrightarrow_{\mathcal{R}_1} 1 + \underline{\mathsf{sum}(1,2)} \hookrightarrow_{\mathcal{R}_1} 1 + \mathsf{sif}(\underline{1 \geqslant 2}, 1, 2) \hookrightarrow_{\mathcal{R}_1} 1 + \underline{\mathsf{sif}(\mathsf{false}, 1, 2)} \\ &\hookrightarrow_{\mathcal{R}_1} \underline{1+0} \hookrightarrow_{\mathcal{R}_1} 1 \end{aligned}$$

3 Integer Dependency Pair Framework

The *DP framework* [1,12,13,16,19] is one of the most powerful and popular methods for automated termination analysis of TRSs and the DP technique is implemented in almost all current TRS termination tools. Our goal is to extend the DP framework in order to handle ITRSs. The main problem is that proving innermost termination of $\mathcal{R} \cup \mathcal{PD}$ *automatically* is not straightforward, as the TRS $\mathcal{PD}$ is infinite. Therefore, we will not consider the rules $\mathcal{PD}$ explicitly, but integrate their handling in the different processors of the DP framework instead.

Of course, the resulting method should be as powerful as possible for term rewriting on integers, but at the same time it should have the full power of the original DP framework when dealing with other function symbols. In particular, if an ITRS does not contain any symbols from $\mathcal{F}_{int}$, then our new variant of the DP framework coincides with the existing DP framework for ordinary TRSs.

As usual, the *defined* symbols $\mathcal{D}$ are the root symbols of left-hand sides of rules. All other symbols are *constructors*. For an ITRS $\mathcal{R}$, we consider all rules in $\mathcal{R} \cup \mathcal{PD}$ to determine the defined symbols, i.e., here $\mathcal{D}$ also includes $\mathcal{A}rith\mathcal{O}p \cup \mathcal{R}el\mathcal{O}p \cup \mathcal{B}ool\mathcal{O}p$. Nevertheless, we ignore these symbols when building DPs, since these DPs would never be the reason for non-termination.[5]

Definition 2 (DP). *For all $f \in \mathcal{D} \setminus \mathcal{F}_{int}$, we introduce a fresh* tuple symbol *$f^\sharp$ with the same arity, where we often write F instead of $f^\sharp$. If $t = f(t_1, ..., t_n)$, let $t^\sharp = f^\sharp(t_1, ..., t_n)$. If $\ell \to r \in \mathcal{R}$ for an ITRS $\mathcal{R}$ and t is a subterm of r with* root$(t) \in \mathcal{D} \setminus \mathcal{F}_{int}$*, then $\ell^\sharp \to t^\sharp$ is a* dependency pair *of $\mathcal{R}$. $DP(\mathcal{R})$ is the set of all DPs.*

For example, we have $DP(\mathcal{R}_1) = \{(7), (8)\}$ and $DP(\mathcal{R}_2) = \{(9), (10)\}$, where

$$\mathsf{SUM}(x,y) \to \mathsf{SIF}(x \geqslant y, x, y) \quad (7) \qquad \mathsf{LOG}(x,y) \to \mathsf{LIF}(x \geqslant y \wedge y > 1, x, y) \quad (9)$$
$$\mathsf{SIF}(\mathsf{true}, x, y) \to \mathsf{SUM}(x, y+1) \quad (8) \qquad \mathsf{LIF}(\mathsf{true}, x, y) \to \mathsf{LOG}(x/y, y) \quad (10)$$

The main result of the DP method for innermost termination states that a TRS $\mathcal{R}$ is innermost terminating iff there is no infinite innermost $DP(\mathcal{R})$*-chain*.

[5] Formally, they would never occur in any infinite *chain* and could easily be removed by standard techniques like the so-called *dependency graph* [1,12].

This can be adapted to ITRSs in a straightforward way. For any TRS $\mathcal{P}$ and ITRS $\mathcal{R}$, a *$\mathcal{P}$-chain* is a sequence of variable renamed pairs $s_1 \to t_1, s_2 \to t_2, \ldots$ from $\mathcal{P}$ such that there is a substitution σ (with possibly infinite domain) where $t_i\sigma \hookrightarrow_{\mathcal{R}}^* s_{i+1}\sigma$ and $s_i\sigma$ is in normal form w.r.t. $\hookrightarrow_{\mathcal{R}}$, for all i. Then we immediately get the following corollary from the standard results on DPs.[6]

Corollary 3 (Termination Criterion for ITRSs). *An ITRS $\mathcal{R}$ is terminating (w.r.t. $\hookrightarrow_{\mathcal{R}}$) iff there is no infinite $DP(\mathcal{R})$-chain.*

Termination techniques are now called *DP processors* and they operate on sets of DPs (called *DP problems*).[7] A DP processor *Proc* takes a DP problem as input and returns a set of new DP problems which have to be solved instead. *Proc* is *sound* if for all DP problems $\mathcal{P}$ with an infinite $\mathcal{P}$-chain there is also a $\mathcal{P}' \in Proc(\mathcal{P})$ with an infinite $\mathcal{P}'$-chain. Soundness of processors is required to prove termination and to conclude that there is no infinite $\mathcal{P}$-chain if $Proc(\mathcal{P}) = \varnothing$.

So termination proofs in the DP framework start with the initial DP problem $DP(\mathcal{R})$. Then the DP problem is simplified repeatedly by sound DP processors. If all resulting DP problems have been simplified to $\varnothing$, then termination is proved. Many processors (like the well-known *(estimated) dependency graph processor* [1,12,13], for example) do not rely on the rules of the TRS, but just on the DPs and on the defined symbols. Therefore, they can also be directly applied for ITRSs, since the sets of DPs and of defined symbols are finite and one does not have to consider the infinitely many rules in $\mathcal{PD}$. One just has to take into account that the defined symbols also include $\mathcal{A}rith\mathcal{O}p \cup \mathcal{R}el\mathcal{O}p \cup \mathcal{B}ool\mathcal{O}p$.

But an adaption is non-trivial for one of the most important processors, the *reduction pair processor*. Thus, the main contribution of the paper is to adapt this processor to obtain a powerful automated termination method for ITRSs.

For a DP problem $\mathcal{P}$, the reduction pair processor generates constraints which should be satisfied by a suitable order on terms. In this paper, we consider orders based on *integer*[8] *max-polynomial interpretations* [11,17]. Such interpretations suffice for most algorithms typically occurring in practice. The set of *max-polynomials* is the smallest set containing the integers $\mathbb{Z}$, the variables, and $p + q$, $p * q$, and $\max(p, q)$ for all max-polynomials p and q. An *integer max-polynomial interpretation* $\mathcal{P}ol$ maps every[9] n-ary function symbol f to a max-polynomial

[6] For Cor. 3, it suffices to consider only *minimal* chains where all $t_i\sigma$ are $\hookrightarrow_{\mathcal{R}}$-terminating [13]. All results of this paper also hold for *minimal* instead of ordinary chains.

[7] To ease readability we use a simpler definition of *DP problems* than [13], since this simple definition suffices for the presentation of the new results of this paper.

[8] Interpretations into the *integers* instead of the naturals are often needed for algorithms like sum that *increase* an argument y until it reaches a *bound* x. In [17], we already presented an approach to prove termination by *bounded increase*. However, [17] did not consider built-in integers and pre-defined operations on them. Instead, [17] only handled natural numbers and all operations (like "$\geqslant$") had to be defined by rules of the TRS itself. Therefore, we now extend the approach of [17] to ITRSs.

[9] This is more general than related previous classes of interpretations: In [17], there was no "max" and only tuple symbols could be mapped to polynomials with integer coefficients, and in [11], all ground terms had to be mapped to natural numbers.

$f_{\mathcal{P}ol}$ over n variables $x_1, \ldots, x_n$. This mapping is extended to terms by defining $[x]_{\mathcal{P}ol} = x$ for all variables x and by letting $[f(t_1, \ldots, t_n)]_{\mathcal{P}ol} = f_{\mathcal{P}ol}([t_1]_{\mathcal{P}ol}, \ldots, [t_n]_{\mathcal{P}ol})$. One now defines $s \succ_{\mathcal{P}ol} t$ (resp. $s \succsim_{\mathcal{P}ol} t$) iff $[s]_{\mathcal{P}ol} > [t]_{\mathcal{P}ol}$ (resp. $[s]_{\mathcal{P}ol} \geqslant [t]_{\mathcal{P}ol}$) holds for all instantiations of the variables with integer numbers.

For example, consider the interpretation $\mathcal{P}ol$ where $\mathsf{SUM}_{\mathcal{P}ol} = x_1 - x_2$, $\mathsf{SIF}_{\mathcal{P}ol} = x_2 - x_3$, $+_{\mathcal{P}ol} = x_1 + x_2$, $n_{\mathcal{P}ol} = n$ for all $n \in \mathbb{Z}$, and $\geqslant_{\mathcal{P}ol} = \mathsf{true}_{\mathcal{P}ol} = \mathsf{false}_{\mathcal{P}ol} = 0$. For any term t and any position π in t, we say that t is $\succsim_{\mathcal{P}ol}$-*dependent* on π iff there exist terms u, v where $t[u]_\pi \not\approx_{\mathcal{P}ol} t[v]_\pi$. Here, $\approx_{\mathcal{P}ol} = \succsim_{\mathcal{P}ol} \cap \precsim_{\mathcal{P}ol}$. So in our example, $\mathsf{SIF}(b, x, y)$ is $\succsim_{\mathcal{P}ol}$-dependent on 2 and 3, but not on 1. We say that a term t is $\succsim_{\mathcal{P}ol}$-*increasing* on π iff $u \succsim_{\mathcal{P}ol} v$ implies $t[u]_\pi \succsim_{\mathcal{P}ol} t[v]_\pi$ for all terms u, v. So clearly, if t is $\succsim_{\mathcal{P}ol}$-independent on π, then t is also $\succsim_{\mathcal{P}ol}$-increasing on π. In our example, $\mathsf{SIF}(b, x, y)$ is $\succsim_{\mathcal{P}ol}$-increasing on 1 and 2, but not on 3.

The constraints generated by the reduction pair processor require that all DPs in $\mathcal{P}$ are strictly or weakly decreasing and all *usable rules* are weakly decreasing. Then one can delete all strictly decreasing DPs.

The *usable rules* [1,16] include all rules that can reduce terms in $\succsim_{\mathcal{P}ol}$-dependent positions of $\mathcal{P}$'s right-hand sides when instantiating their variables with normal forms. Formally, for a term with f on a $\succsim_{\mathcal{P}ol}$-dependent position, all f-rules are usable. Moreover, if f's rules are usable and g occurs in the right-hand side of an f-rule on a $\succsim_{\mathcal{P}ol}$-dependent position, then g's rules are usable as well. For any symbol f with $\mathrm{arity}(f) = n$, let $dep(f) = \{i \mid 1 \leqslant i \leqslant n$, there exists a term $f(t_1, \ldots, t_n)$ that is $\succsim_{\mathcal{P}ol}$-dependent on $i\}$. So $dep(\mathsf{SIF}) = \{2, 3\}$ for the interpretation $\mathcal{P}ol$ above. Moreover, as $\succsim_{\mathcal{P}ol}$ is not monotonic in general, one has to require that defined symbols only occur on $\succsim_{\mathcal{P}ol}$-increasing positions of right-hand sides.[10]

When using interpretations into the integers, then $\succ_{\mathcal{P}ol}$ is not well founded. However, $\succ_{\mathcal{P}ol}$ is still "non-infinitesimal", i.e., for any given bound, there is no infinite $\succ_{\mathcal{P}ol}$-decreasing sequence of terms that remains greater than the bound. Hence, the reduction pair processor transforms a DP problem into *two* new problems. As mentioned before, the first problem results from removing all strictly decreasing DPs. The second DP problem results from removing all DPs $s \to t$ from $\mathcal{P}$ that are *bounded from below*, i.e., DPs which satisfy the inequality $s \succsim \mathsf{c}$ for a fresh constant c. In Thm. 4, both TRSs and relations are seen as sets of pairs of terms. Thus, "$\mathcal{P} \setminus \succ_{\mathcal{P}ol}$" denotes $\{s \to t \in \mathcal{P} \mid s \not\succ_{\mathcal{P}ol} t\}$. Moreover, for any function symbol f and any TRS $\mathcal{S}$, let $Rls_{\mathcal{S}}(f) = \{\ell \to r \in \mathcal{S} \mid \mathrm{root}(\ell) = f\}$.

Theorem 4 (Reduction Pair Processor [17]). *Let $\mathcal{R}$ be an ITRS, $\mathcal{P}ol$ be an integer max-polynomial interpretation, c be a fresh constant, and $\mathcal{P}_{bound} = \{s \to t \in \mathcal{P} \mid s \succsim_{\mathcal{P}ol} \mathsf{c}\}$. Then the following DP processor Proc is sound.*

[10] This is needed to ensure that $t\sigma \hookrightarrow^*_{\mathcal{R}} u$ implies $t\sigma \succsim_{\mathcal{P}ol} u$ whenever t's usable rules are weakly decreasing and σ instantiates variables by normal forms. Note that Thm. 4 is a simplified special case of the corresponding processor from [17]. In [17], we also introduced the possibility of reversing usable rules for function symbols occurring on *decreasing* positions. The approach of the present paper can also easily be extended accordingly and, indeed, our implementation makes use of this extension.

$$Proc(\mathcal{P}) = \begin{cases} \{\, \mathcal{P} \setminus \succ_{\mathcal{P}ol},\ \mathcal{P} \setminus \mathcal{P}_{bound} \,\}, & \text{if } \mathcal{P} \subseteq \gtrsim_{\mathcal{P}ol} \cup \succ_{\mathcal{P}ol},\ \mathcal{U}_{\mathcal{R}\cup\mathcal{PD}}(\mathcal{P}) \subseteq \gtrsim_{\mathcal{P}ol}, \\ & \text{and defined symbols only occur on} \\ & \gtrsim_{\mathcal{P}ol}\text{-increasing positions} \\ & \text{in right-hand sides of } \mathcal{P} \cup \mathcal{U}_{\mathcal{R}}(\mathcal{P}) \\ \{\, \mathcal{P} \,\}, & \text{otherwise} \end{cases}$$

For any term t and TRS $\mathcal{S}$, the usable rules $\mathcal{U}_{\mathcal{S}}(t)$ *are the smallest set with*

- $\mathcal{U}_{\mathcal{S}}(x) = \varnothing$ *for every variable x and*
- $\mathcal{U}_{\mathcal{S}}(f(t_1, \ldots, t_n)) = Rls_{\mathcal{S}}(f) \cup \bigcup_{\ell\to r\in Rls_{\mathcal{S}}(f)} \mathcal{U}_{\mathcal{S}}(r) \cup \bigcup_{i\in dep(f)} \mathcal{U}_{\mathcal{S}}(t_i)$

For a set of dependency pairs $\mathcal{P}$, its usable rules *are* $\mathcal{U}_{\mathcal{S}}(\mathcal{P}) = \bigcup_{s\to t\in\mathcal{P}} \mathcal{U}_{\mathcal{S}}(t)$.

For $\mathcal{R}_1$, by Thm. 4 we search for an interpretation $\mathcal{P}ol$ with $s \gtrsim_{(\mathcal{P}ol)} t$ for all $s \to t \in DP(\mathcal{R}_1) = \{(7), (8)\}$ and $\ell \gtrsim_{\mathcal{P}ol} r$ for all $\ell \to r \in \mathcal{U}_{\mathcal{R}_1\cup\mathcal{PD}}(DP(\mathcal{R}_1)) = \{0+1 \to 1,\ 1+1 \to 2,\ -1+1 \to 0,\ \ldots,\ 0 \geqslant 0 \to \mathsf{true},\ 1 \geqslant 2 \to \mathsf{false},\ \ldots\}$. However, $\mathcal{U}_{\mathcal{R}_1\cup\mathcal{PD}}(DP(\mathcal{R}_1))$ is infinite and thus, this approach is not suitable for automation. When using the interpretation with $\mathsf{SIF}_{\mathcal{P}ol} = x_2 - x_3$, then the $\geqslant$-rules would not be usable, because $\geqslant$ only occurs on a $\gtrsim_{\mathcal{P}ol}$-independent position in the right-hand side of the DP (7). But the desired interpretation $\mathsf{SUM}_{\mathcal{P}ol} = x_1 - x_2$ cannot be used, because in DP (8), the defined symbol $+$ occurs in the second argument of SUM which is not a $\gtrsim_{\mathcal{P}ol}$-increasing position.[11]

To avoid the need for considering infinitely many rules in the reduction pair processor and in order to handle ITRSs where defined symbols like $+$ occur on non-increasing positions, we will now restrict ourselves to so-called *I-interpretations* where we fix the max-polynomials that are associated with the pre-defined symbols from $\mathbb{Z} \cup \mathcal{A}rith\mathcal{O}p$. The definition of I-interpretations guarantees that we have $\ell \approx_{\mathcal{P}ol} r$ for all rules $\ell \to r \in \mathcal{PD}$ where $\mathrm{root}(\ell) \in \{+, -, *\}$. For this reason, one can now also allow occurrences of $+$, $-$, and $*$ on non-increasing positions. Moreover, for I-interpretations we have $\ell \gtrsim_{\mathcal{P}ol} r$ for all rules $\ell \to r \in \mathcal{PD}$ where $\mathrm{root}(\ell) \in \{/, \%\}$. For these latter rules, obtaining $\ell \approx_{\mathcal{P}ol} r$ with a useful max-polynomial interpretation is impossible, since division and modulo are no max-polynomials.[12]

Definition 5 (I-interpretation). *An integer max-polynomial interpretation $\mathcal{P}ol$ is an* I-interpretation *iff $n_{\mathcal{P}ol} = n$ for all $n \in \mathbb{Z}$, $+_{\mathcal{P}ol} = x_1 + x_2$, $-_{\mathcal{P}ol} =$*

[11] Nevertheless, Thm. 4 is helpful for ITRSs where the termination argument is not due to integer arithmetic. For example, consider the ITRS $\mathsf{g}(x, \mathsf{cons}(y, ys)) \to \mathsf{cons}(x + y, \mathsf{g}(x, ys))$. When using interpretations $\mathcal{P}ol$ with $f_{\mathcal{P}ol} = 0$ for all $f \in \mathcal{F}_{int}$, the rules $\ell \to r \in \mathcal{PD}$ are always weakly decreasing. Hence, then one only has to regard finitely many usable rules when automating Thm. 4. Moreover, if all $f_{\mathcal{P}ol}$ have just *natural* coefficients, then one does not have to generate the new DP problem $\mathcal{P} \setminus \mathcal{P}_{bound}$. In this case, one can define $s \gtrsim_{(\mathcal{P}ol)} t$ iff $[s]_{\mathcal{P}ol}\ (\geqslant)\ [t]_{\mathcal{P}ol}$ holds for all instantiations of the variables by *natural* numbers. Thus, in the example above the termination proof is trivial by using the interpretation with $\mathsf{G}_{\mathcal{P}ol} = x_2$ and $\mathsf{cons}_{\mathcal{P}ol} = x_2 + 1$.

[12] In principle, one could also permit interpretations $f_{\mathcal{P}ol}$ containing divisions. But existing implementations to search for interpretations cannot handle division or modulo.

$x_1 - x_2$, $*_{\mathcal{P}ol} = x_1 * x_2$, $\%_{\mathcal{P}ol} = |x_1|$, *and* $/_{\mathcal{P}ol} = |x_1| - \min(|x_2| - 1, |x_1|)$. *Note that for any max-polynomial* p, "$|p|$" *is also a max-polynomial since this is just an abbreviation for* $\max(p, -p)$. *Similarly, "*$\min(p, q)$*" is an abbreviation for* $-\max(-p, -q)$. *We say that an I-interpretation is* proper *for a term* t *if all defined symbols except* $+$, $-$, *and* $*$ *only occur on* $\succsim_{\mathcal{P}ol}$*-increasing positions of* t *and if symbols from* $\mathcal{R}el\mathcal{O}p$ *only occur on* $\succsim_{\mathcal{P}ol}$*-independent positions of* t.

Now $[n/m]_{\mathcal{P}ol}$ is greater or equal to n/m for all $n, m \in \mathbb{Z}$ where $m \neq 0$ (and similar for $[n \,\%\, m]_{\mathcal{P}ol}$).[13] Hence, one can improve the processor of Thm. 4 by not regarding the infinitely many rules of $\mathcal{PD}$ anymore. The concept of *proper* I-interpretations ensures that we can disregard the (infinitely many) usable rules for the symbols from $\mathcal{R}el\mathcal{O}p$ and that the symbols "/" and "%" only have to be estimated "upwards". Then one can now replace the usable rules w.r.t. $\mathcal{R} \cup \mathcal{PD}$ in Thm. 4 by the usable rules w.r.t. $\mathcal{R} \cup \mathcal{BO}$. Here, $\mathcal{BO}$ are the (finitely many) rules for the symbols $\wedge$ and $\Rightarrow$ in $\mathcal{B}ool\mathcal{O}p$, i.e., $\mathcal{BO} = Rls_{\mathcal{PD}}(\wedge) \cup Rls_{\mathcal{PD}}(\Rightarrow)$.

Theorem 6 (Reduction Pair Processor for ITRSs). *Let* $\mathcal{R}$ *be an ITRS,* $\mathcal{P}ol$ *be an I-interpretation, and* $\mathcal{P}_{bound}$ *be as in Thm. 4. Then Proc is sound.*

$$Proc(\mathcal{P}) = \begin{cases} \{\, \mathcal{P} \setminus \succ_{\mathcal{P}ol},\ \mathcal{P} \setminus \mathcal{P}_{bound} \,\}, & \text{if } \mathcal{P} \subseteq \succsim_{\mathcal{P}ol} \cup \succ_{\mathcal{P}ol},\ \mathcal{U}_{\mathcal{R} \cup \mathcal{BO}}(\mathcal{P}) \subseteq \succsim_{\mathcal{P}ol}, \\ & \text{and } \mathcal{P}ol \text{ is proper for all right-hand} \\ & \text{sides of } \mathcal{P} \cup \mathcal{U}_{\mathcal{R}}(\mathcal{P}) \\ \{\, \mathcal{P} \,\}, & \text{otherwise} \end{cases}$$

Proof. We show that Thm. 6 follows from Thm. 4. In Thm. 6, we only require that usable rules from $\mathcal{R} \cup \mathcal{BO}$ are weakly decreasing, whereas Thm. 4 considers usable rules from $\mathcal{R} \cup \mathcal{PD}$. For any I-interpretation $\mathcal{P}ol$, we have $\ell \approx_{\mathcal{P}ol} r$ for all $\ell \to r \in \mathcal{PD}$ with $\mathrm{root}(\ell) \in \{+, -, *\}$. So these rules are even equivalent w.r.t. $\approx_{\mathcal{P}ol}$. Moreover, the rules with $\mathrm{root}(\ell) \in \{/, \%\}$ are weakly decreasing w.r.t. $\succsim_{\mathcal{P}ol}$. The rules with $\mathrm{root}(\ell) \in \mathcal{R}el\mathcal{O}p$ are never contained in $\mathcal{U}_{\mathcal{R} \cup \mathcal{PD}}(\mathcal{P})$, because by properness of $\mathcal{P}ol$, symbols from $\mathcal{R}el\mathcal{O}p$ only occur on $\succsim_{\mathcal{P}ol}$-independent positions in right-hand sides of $\mathcal{P} \cup \mathcal{U}_{\mathcal{R}}(\mathcal{P})$ and they do not occur at all in right-hand sides of $\mathcal{PD}$. Thus, $\mathcal{U}_{\mathcal{R} \cup \mathcal{PD}}(\mathcal{P}) \subseteq \succsim_{\mathcal{P}ol}$, as required in Thm. 4.

The other difference between Thm. 6 and 4 is that in Thm. 6, $+$, $-$, and $*$ may also occur on non-$\succsim_{\mathcal{P}ol}$-increasing positions. But as shown in [17,20], this is possible since the rules for these symbols are equivalent w.r.t. $\approx_{\mathcal{P}ol}$. □

To solve the DP problem $\mathcal{P} = \{(7), (8)\}$ of $\mathcal{R}_1$ with Thm. 6, we want to use an I-interpretation $\mathcal{P}ol$ where $\mathsf{SUM}_{\mathcal{P}ol} = x_1 - x_2$ and $\mathsf{SIF}_{\mathcal{P}ol} = x_2 - x_3$. Now there are no usable rules $\mathcal{U}_{\mathcal{R} \cup \mathcal{BO}}(\mathcal{P})$, since the $+$- and $\geqslant$-rules are not included in $\mathcal{R} \cup \mathcal{BO}$. The DP (8) is strictly decreasing, but none of the DPs (7) and (8) is bounded, since we have neither $\mathsf{SUM}(x, y) \succsim_{\mathcal{P}ol} \mathsf{c}$ nor $\mathsf{SIF}(\mathsf{true}, x, y) \succsim_{\mathcal{P}ol} \mathsf{c}$ for any possible value of $\mathsf{c}_{\mathcal{P}ol}$. Thus, the reduction pair processor would return the two DP problems $\{(7)\}$ and $\{(7), (8)\}$, i.e., it would not simplify $\mathcal{P}$.

[13] Let $m \neq 0$. If $|m| = 1$ or $n = 0$, then we have $[n/m]_{\mathcal{P}ol} = |n|$. Otherwise, we obtain $[n/m]_{\mathcal{P}ol} < |n|$. The latter fact is needed for ITRSs like $\mathcal{R}_2$ which terminate because of divisions in their recursive arguments.

4 Conditional Constraints

The solution to the problem above is to consider *conditions* for inequalities like $s \mathrel{(\gtrsim)} t$ or $s \gtrsim \mathsf{c}$. For example, to include the DP (7) in $\mathcal{P}_{bound}$, we do not have to demand $\mathsf{SUM}(x,y) \gtrsim \mathsf{c}$ for *all* instantiations of x and y. Instead, it suffices to require the inequality only for those instantiations of x and y which can be used in chains. So we require $\mathsf{SUM}(x,y) \gtrsim \mathsf{c}$ only for instantiations σ where (7)'s instantiated right-hand side $\mathsf{SIF}(x \geqslant y, x, y)\sigma$ reduces to an instantiated left-hand side $u\sigma$ for some DP $u \to v$ where $u\sigma$ is in normal form. Here, $u \to v$ should again be variable renamed. As our DP problem contains two DPs (7) and (8), we get the following two *conditional constraints* (by considering all possibilities $u \to v \in \{(7),(8)\}$). We include (7) in $\mathcal{P}_{bound}$ if both constraints are satisfied.

$$\mathsf{SIF}(x \geqslant y, x, y) = \mathsf{SUM}(x', y') \quad \Rightarrow \quad \mathsf{SUM}(x,y) \gtrsim \mathsf{c} \tag{11}$$
$$\mathsf{SIF}(x \geqslant y, x, y) = \mathsf{SIF}(\mathsf{true}, x', y') \quad \Rightarrow \quad \mathsf{SUM}(x,y) \gtrsim \mathsf{c} \tag{12}$$

Definition 7 (Syntax and Semantics of Conditional Constraints [17]). *The set $\mathcal{C}$ of conditional constraints is the smallest set with*[14]

- $\{\mathit{TRUE},\ s \gtrsim t,\ s \succ t,\ s = t\} \subseteq \mathcal{C}$ *for all terms s and t*
- *if $\{\varphi_1, \varphi_2\} \subseteq \mathcal{C}$, then $\varphi_1 \wedge \varphi_2 \in \mathcal{C}$ and $\varphi_1 \Rightarrow \varphi_2 \in \mathcal{C}$*

For an I-interpretation $\mathcal{P}ol$, we define which normal substitutions[15] *σ satisfy a constraint $\varphi \in \mathcal{C}$, denoted "$\sigma \models_{\mathcal{P}ol} \varphi$":*

- $\sigma \models_{\mathcal{P}ol} \mathit{TRUE}$ *for all normal substitutions σ*
- $\sigma \models_{\mathcal{P}ol} s \gtrsim t$ *iff* $s\sigma \gtrsim_{\mathcal{P}ol} t\sigma$ *and* $\sigma \models_{\mathcal{P}ol} s \succ t$ *iff* $s\sigma \succ_{\mathcal{P}ol} t\sigma$
- $\sigma \models_{\mathcal{P}ol} s = t$ *iff* $s\sigma \hookrightarrow_{\mathcal{R}}^* t\sigma$ *and $t\sigma$ is a normal form w.r.t.* $\hookrightarrow_{\mathcal{R}}$
- $\sigma \models_{\mathcal{P}ol} \varphi_1 \wedge \varphi_2$ *iff* $\sigma \models_{\mathcal{P}ol} \varphi_1$ *and* $\sigma \models_{\mathcal{P}ol} \varphi_2$
- $\sigma \models_{\mathcal{P}ol} \varphi_1 \Rightarrow \varphi_2$ *iff* $\sigma \not\models_{\mathcal{P}ol} \varphi_1$ *or* $\sigma \models_{\mathcal{P}ol} \varphi_2$

A constraint φ is valid *("$\models_{\mathcal{P}ol} \varphi$") iff $\sigma \models_{\mathcal{P}ol} \varphi$ for* all *normal substitutions σ.*

Now we refine the reduction pair processor by taking conditions into account. To this end, we modify the definition of $\mathcal{P}_{bound}$ and introduce $\mathcal{P}_{\gtrsim}$ and $\mathcal{P}_{\succ}$.

Theorem 8 (Conditional Reduction Pair Processor for ITRSs). *Let $\mathcal{R}$ be an ITRS, $\mathcal{P}ol$ be an I-interpretation, c be a fresh constant, and let*

$$\mathcal{P}_{\gtrsim} = \{\ s \to t \in \mathcal{P} \ \mid\ \models_{\mathcal{P}ol} \textstyle\bigwedge_{u \to v \in \mathcal{P}} (t = u' \ \Rightarrow\ s \gtrsim t)\ \}$$
$$\mathcal{P}_{\succ} = \{\ s \to t \in \mathcal{P} \ \mid\ \models_{\mathcal{P}ol} \textstyle\bigwedge_{u \to v \in \mathcal{P}} (t = u' \ \Rightarrow\ s \succ t)\ \}$$
$$\mathcal{P}_{bound} = \{\ s \to t \in \mathcal{P} \ \mid\ \models_{\mathcal{P}ol} \textstyle\bigwedge_{u \to v \in \mathcal{P}} (t = u' \ \Rightarrow\ s \gtrsim \mathsf{c})\ \}$$

[14] To simplify the presentation, we neither regard conditional constraints with universally quantified subformulas nor the simplification of constraints by induction, cf. [17]. This technique of [17] could be integrated in our approach to also handle ITRSs where tests are not of the form "$s \geqslant t$" with the pre-defined symbol "$\geqslant$", but of the form "$\mathsf{ge}(s,t)$", where ge is given by user-defined rules in the ITRS. After such an integration, our approach would subsume the corresponding technique of [17].

[15] A *normal substitution* σ instantiates all variables by normal forms w.r.t. $\hookrightarrow_{\mathcal{R}}$.

where u' results from u by renaming its variables. Then Proc is sound.

$$Proc(\mathcal{P}) = \begin{cases} \{\, \mathcal{P} \setminus \mathcal{P}_{\succ},\ \mathcal{P} \setminus \mathcal{P}_{bound} \,\}, & \text{if } \mathcal{P}_{\gtrsim} \cup \mathcal{P}_{\succ} = \mathcal{P},\ \mathcal{U}_{\mathcal{R} \cup \mathcal{BO}}(\mathcal{P}) \subseteq \gtrsim_{\mathcal{P}ol}, \\ & \text{and } \mathcal{P}ol \text{ is proper for all right-hand} \\ & \text{sides of } \mathcal{P} \cup \mathcal{U}_{\mathcal{R}}(\mathcal{P}) \\ \{\, \mathcal{P} \,\}, & \text{otherwise} \end{cases}$$

Proof. Thm. 8 immediately follows from Thm. 6 in the same way as [17, Thm. 11] follows from [17, Thm. 8]. □

To ease readability, in Thm. 8 we only consider conditions resulting from *two* DPs $s \to t$ and $u \to v$ which may follow each other in chains. In our implementation, we extended this by also regarding conditions resulting from more than two adjacent DPs and by also regarding DPs *preceding* $s \to t$ in chains, cf. [17].

The question remains how to check whether conditional constraints are valid, since this requires reasoning about reachability w.r.t. TRSs with infinitely many rules. In [17], we introduced the rules (I)-(IV) to simplify conjunctions $\varphi_1 \wedge ... \wedge \varphi_n$ of conditional constraints. These rules can be used to replace a conjunct φ_i by a new formula φ_i'. The rules are sound, i.e., $\models_{\mathcal{P}ol} \varphi_i'$ implies $\models_{\mathcal{P}ol} \varphi_i$. Of course, $TRUE \wedge \varphi$ can always be simplified to φ. Eventually, we want to remove all equalities "$p = q$" from the constraints.

I. **Constructor and Different Function Symbol**

$$\frac{f(s_1, ..., s_n) = g(t_1, ..., t_m) \wedge \varphi \ \Rightarrow\ \psi}{TRUE} \quad \text{if } f \text{ is a constructor and } f \neq g$$

II. **Same Constructors on Both Sides**

$$\frac{f(s_1, ..., s_n) = f(t_1, ..., t_n) \wedge \varphi \ \Rightarrow\ \psi}{s_1 = t_1 \wedge \ldots \wedge s_n = t_n \wedge \varphi \ \Rightarrow\ \psi} \quad \text{if } f \text{ is a constructor}$$

III. **Variable in Equation**

$$\frac{x = q \wedge \varphi \ \Rightarrow\ \psi}{\varphi\sigma \ \Rightarrow\ \psi\sigma} \quad \text{if } x \in \mathcal{V} \text{ and } \sigma = [x/q] \qquad \frac{q = x \wedge \varphi \ \Rightarrow\ \psi}{\varphi\sigma \ \Rightarrow\ \psi\sigma} \quad \text{if } x \in \mathcal{V},\ q \text{ has no defined symbols, } \sigma = [x/q]$$

IV. **Delete Conditions**

$$\frac{\varphi \ \Rightarrow\ \psi}{\varphi' \ \Rightarrow\ \psi} \quad \text{if } \varphi' \subseteq \varphi$$

For example, Rule (I) detects that the premise of constraint (11) is unsatisfiable: there is no substitution σ with $\sigma \models_{\mathcal{P}ol} \mathsf{SIF}(x \geqslant y, x, y) = \mathsf{SUM}(x', y')$, since SIF is not a defined function symbol (i.e., it is a *constructor*) and therefore, SIF-terms can only be reduced to SIF-terms.

Rule (II) handles conditions like $\mathsf{SIF}(x \geqslant y, x, y) = \mathsf{SIF}(\mathsf{true}, x', y')$ where both terms start with the same constructor SIF. So (12) is transformed into

$$x \geqslant y = \mathsf{true} \ \wedge\ x = x' \ \wedge\ y = y' \quad \Rightarrow \quad \mathsf{SUM}(x, y) \gtrsim \mathsf{c} \tag{13}$$

Rule (III) removes conditions of the form "$x = q$" or "$q = x$" by applying the substitution $[x/q]$ to the constraint. So (13) is transformed into

$$x \geqslant y = \mathsf{true} \quad \Rightarrow \quad \mathsf{SUM}(x, y) \gtrsim \mathsf{c} \tag{14}$$

Rule (IV) can omit arbitrary conjuncts from the premise of an implication. To ease notation, we regard a conjunction as a set of formulas. So their order is irrelevant and we write $\varphi' \subseteq \varphi$ iff all conjuncts of φ' are also conjuncts of φ. The empty conjunction is TRUE (i.e., $\mathit{TRUE} \Rightarrow \psi$ can always be simplified to ψ).

Since [17] did not handle pre-defined function symbols, we now extend the rules (I)-(IV) from [17] by new rules to "lift" pre-defined function symbols from $\mathcal{R}el\mathcal{O}p$ like $\geqslant$ to symbols like $\gtrsim$ that are used in conditional constraints. Similar rules are used for the other symbols from $\mathcal{R}el\mathcal{O}p$. The idea is to replace a conditional constraint like "$s \geqslant t = \mathsf{true}$" by the conditional constraint "$s \gtrsim t$". However, this is not necessarily sound, because s and t may contain defined symbols. Note that $\sigma \models_{\mathcal{P}ol} s \geqslant t = \mathsf{true}$ means that $s\sigma \hookrightarrow^*_{\mathcal{R}} n$ and $t\sigma \hookrightarrow^*_{\mathcal{R}} m$ for $n, m \in \mathbb{Z}$ with $n \geqslant m$. For any I-interpretation $\mathcal{P}ol$, we therefore have $n \gtrsim_{\mathcal{P}ol} m$, since $n_{\mathcal{P}ol} = n$ and $m_{\mathcal{P}ol} = m$. To guarantee that $\sigma \models_{\mathcal{P}ol} s \gtrsim t$ holds as well, we ensure that $s\sigma \gtrsim_{\mathcal{P}ol} n$ and $m \gtrsim_{\mathcal{P}ol} t\sigma$. To this end, we require that $\mathcal{U}_{\mathcal{R} \cup \mathcal{BO}}(s) \subseteq \gtrsim_{\mathcal{P}ol}$ and that t contains no defined symbols except $+$, $-$, and $*$. An analogous rule can also be formulated for constraints of the form "$s \geqslant t = \mathsf{false}$".[16]

V. **Lift Symbols from** $\mathcal{R}el\mathcal{O}p$

$$\frac{s \geqslant t = \mathsf{true} \wedge \varphi \;\Rightarrow\; \psi}{(s \gtrsim t \quad \wedge \varphi \;\Rightarrow\; \psi) \wedge \bigwedge_{\ell \to r \in \mathcal{U}_{\mathcal{R} \cup \mathcal{BO}}(s)} \ell \gtrsim r}$$

if t contains no defined symbols except $+$, $-$, $*$ and $\mathcal{P}ol$ is proper for s and for all right-hand sides of $\mathcal{U}_{\mathcal{R}}(s)$

By Rule (V), (14) is transformed into

$$x \gtrsim y \quad \Rightarrow \quad \mathsf{SUM}(x, y) \gtrsim \mathsf{c} \tag{15}$$

Similar to the lifting of the function symbols from $\mathcal{R}el\mathcal{O}p$, it is also possible to lift the function symbols from $\mathcal{B}ool\mathcal{O}p$. For reasons of space, we only present the corresponding rules for lifting "$\wedge$", but of course "$\Rightarrow$" can be lifted analogously.

VI. **Lift Symbols from** $\mathcal{B}ool\mathcal{O}p$

$$\frac{s \wedge t = \mathsf{true} \quad \wedge \varphi \;\Rightarrow\; \psi}{s = \mathsf{true} \wedge t = \mathsf{true} \wedge \varphi \;\Rightarrow\; \psi} \qquad \frac{s \wedge t = \mathsf{false} \wedge \varphi \;\Rightarrow\; \psi}{(s = \mathsf{false} \wedge \varphi \;\Rightarrow\; \psi) \wedge (t = \mathsf{false} \wedge \varphi \;\Rightarrow\; \psi)}$$

To illustrate this rule, consider the constraint "$(x \geqslant y \wedge y > 1) = \mathsf{true} \;\Rightarrow\; \mathsf{LOG}(x, y) \gtrsim \mathsf{c}$" which results when trying to include the DP (9) of the ITRS $\mathcal{R}_2$ in $\mathcal{P}_{bound}$. Here, Rule (VI) gives "$x \geqslant y = \mathsf{true} \wedge y > 1 = \mathsf{true} \;\Rightarrow\; \mathsf{LOG}(x, y) \gtrsim \mathsf{c}$" which is transformed by Rule (V) into "$x \gtrsim y \wedge y \succ 1 \;\Rightarrow\; \mathsf{LOG}(x, y) \gtrsim \mathsf{c}$".

Let $\varphi \vdash \varphi'$ iff φ' results from φ by repeatedly applying the above inference rules. We can now refine the processor from Thm. 8.

[16] In addition, one can also use rules to perform narrowing and rewriting on the terms in conditions, similar to the use of narrowing and rewriting in [16].

Theorem 9 (Conditional Reduction Pair Processor with Inference). *Let $\mathcal{P}ol$ be an I-interpretation and c be a fresh constant. For all $s \to t \in \mathcal{P}$ and all $\psi \in \{ s \gtrsim t,\ s \succ t,\ s \gtrsim \mathsf{c} \}$, let φ_ψ be a constraint with $\bigwedge_{u \to v \in \mathcal{P}} (t = u' \ \Rightarrow\ \psi) \vdash \varphi_\psi$. Here, u' results from u by renaming its variables. Then the processor Proc from Thm. 8 is still sound if we define $\mathcal{P}_{\gtrsim} = \{ s \to t \in \mathcal{P} \mid \models_{\mathcal{P}ol} \varphi_{s \gtrsim t} \}$, $\mathcal{P}_{\succ} = \{ s \to t \in \mathcal{P} \mid \models_{\mathcal{P}ol} \varphi_{s \succ t} \}$, and $\mathcal{P}_{bound} = \{ s \to t \in \mathcal{P} \mid \models_{\mathcal{P}ol} \varphi_{s \gtrsim \mathsf{c}} \}$.*

Proof. It suffices to show the soundness of the rules (I)-(VI): If $\varphi \vdash \varphi'$, then $\models_{\mathcal{P}ol} \varphi'$ implies $\models_{\mathcal{P}ol} \varphi$. Then Thm. 9 immediately follows from Thm. 8.

Soundness of the rules (I)-(IV) was shown in [17, Thm. 14]. For Rule (V), let $\models_{\mathcal{P}ol} (s \gtrsim t \wedge \varphi \ \Rightarrow\ \psi) \wedge \bigwedge_{\ell \to r \in \mathcal{U}_{\mathcal{R} \cup \mathcal{BO}}(s)} \ell \gtrsim r$ and $\sigma \models_{\mathcal{P}ol} s \geqslant t = \mathsf{true} \wedge \varphi$. As explained above, this implies $\sigma \models_{\mathcal{P}ol} s \gtrsim t \wedge \varphi$ and hence, $\sigma \models_{\mathcal{P}ol} \psi$, as desired.

For the first variant of (VI), $\sigma \models_{\mathcal{P}ol} s \wedge t = \mathsf{true}$ iff $s\sigma \hookrightarrow^*_{\mathcal{R}} \mathsf{true}$ and $t\sigma \hookrightarrow^*_{\mathcal{R}} \mathsf{true}$, i.e., $\sigma \models_{\mathcal{P}ol} s = \mathsf{true} \wedge t = \mathsf{true}$. For the second variant, $\sigma \models_{\mathcal{P}ol} s \wedge t = \mathsf{false}$ implies $s\sigma \hookrightarrow^*_{\mathcal{R}} \mathsf{false}$ or $t\sigma \hookrightarrow^*_{\mathcal{R}} \mathsf{false}$, i.e., $\sigma \models_{\mathcal{P}ol} s = \mathsf{false}$ or $\sigma \models_{\mathcal{P}ol} t = \mathsf{false}$. □

5 Generating I-Interpretations

To automate the processor of Thm. 9, we show how to generate an I-interpretation that satisfies a given conditional constraint. This conditional constraint is a conjunction of formulas like $\varphi_{s \gtrsim t}$, $\varphi_{s \succ t}$, and $\varphi_{s \gtrsim \mathsf{c}}$ for DPs $s \to t$ as well as $\ell \gtrsim r$ for usable rules $\ell \to r$. Moreover, one has to ensure that the I-interpretation is chosen in such a way that $\mathcal{P}ol$ is proper for the right-hand sides of the DPs and the usable rules.[17] Compared to our earlier work in [11], the only additional difficulty is that now we really consider arbitrary max-polynomial interpretations over the integers where $[t]_{\mathcal{P}ol}$ can also be negative for any ground term t.

To find I-interpretations automatically, one starts with an *abstract* I-interpretation. It maps each function symbol to a max-polynomial with *abstract* coefficients. In other words, one has to determine the degree and the shape of the max-polynomial, but the actual coefficients are left open. For example, for the ITRS $\mathcal{R}_1$ we could use an abstract I-interpretation $\mathcal{P}ol$ where $\mathsf{SUM}_{\mathcal{P}ol} = a_0 + a_1\, x_1 + a_2\, x_2$, $\mathsf{SIF}_{\mathcal{P}ol} = b_0 + b_1\, x_1 + b_2\, x_2 + b_3\, x_3$, and $\mathsf{c}_{\mathcal{P}ol} = c_0$. Here, a_i, b_i, and c_0 are abstract coefficients. Of course, the interpretation for the symbols in $\mathbb{Z} \cup \mathcal{A}rith\mathcal{O}p$ is fixed as for any I-interpretation (i.e., $+_{\mathcal{P}ol} = x_1 + x_2$, etc.).

After application of the rules in Sect. 4, we have obtained a conditional constraint without the symbol "=". Now we transform the conditional constraint into a so-called *inequality constraint* by replacing all atomic constraints "$s \gtrsim t$" by "$[s]_{\mathcal{P}ol} \geqslant [t]_{\mathcal{P}ol}$" and all atomic constraints "$s \succ t$" by "$[s]_{\mathcal{P}ol} \geqslant [t]_{\mathcal{P}ol} + 1$". For instance, the atomic constraint "$\mathsf{SUM}(x, y) \gtrsim \mathsf{c}$" is transformed into "$a_0 + a_1\, x + a_2\, y \geqslant c_0$". Here, the abstract coefficients a_0, a_1, a_2, c_0 are implicitly existentially quantified and the variables $x, y \in \mathcal{V}$ are universally quantified. In other words, we search for values of the abstract coefficients such that the

[17] The set of usable rules and thus, the given conditional constraint depends on the I-interpretation (that determines which positions are increasing or dependent). Nevertheless, we showed in [11] how to encode such search problems into a single constraint.

inequalities hold *for all* integer numbers x and y. To make this explicit, we add universal quantifiers for the variables from $\mathcal{V}$. More precisely, if our overall inequality constraint has the form $\varphi_1 \wedge \ldots \wedge \varphi_n$, then we now replace each φ_i by "$\forall x_1 \in \mathbb{Z}, \ldots, x_m \in \mathbb{Z}\ \varphi_i$" where $x_1, \ldots, x_m$ are the variables from $\mathcal{V}$ occurring in φ_i. So the conditional constraint (15) is transformed into the inequality constraint

$$\forall x \in \mathbb{Z}, y \in \mathbb{Z} \quad (\, x \geqslant y \quad \Rightarrow \quad a_0 + a_1\, x + a_2\, y \geqslant c_0\,) \tag{16}$$

In general, inequality constraints have the following form where Num_i is $\mathbb{Z}$ or $\mathbb{N}$.

$$\forall x_1 \in Num_1, \ldots, x_m \in Num_m \quad p_1 \geqslant q_1 \wedge \ldots \wedge p_n \geqslant q_n \ \Rightarrow\ p \geqslant q$$

Now our goal is to transform such inequality constraints further into *Diophantine constraints* which do not contain any universally quantified variables $x_1, \ldots, x_m$ anymore. Then one can apply existing methods to search for values of the abstract coefficients that satisfy the Diophantine constraints.

We already developed such transformation rules in [11]. But [11] was restricted to the case where all universally quantified variables range over $\mathbb{N}$, i.e., $Num_1 = ... = Num_m = \mathbb{N}$. Moreover, [11]'s essential rule to eliminate universally quantified variables only works if there are no conditions (i.e., $n = 0$), cf. Rule (C) below. Thus, we extend the transformation rules from [11][18] by the following rule which can be used whenever a condition can be transformed into "$x \geqslant p$" or "$p \geqslant x$" for a polynomial p not containing x. It does not only replace a variable ranging over $\mathbb{Z}$ by one over $\mathbb{N}$, but it also "applies" the condition "$x \geqslant p$" resp. "$p \geqslant x$" and removes it afterwards without increasing the number of constraints.

A. **Eliminating Conditions**

$$\frac{\forall x \in \mathbb{Z}, \ldots \quad (x \geqslant p \wedge \varphi \Rightarrow \psi)}{\forall z \in \mathbb{N}, \ldots \quad (\varphi[x/p+z] \Rightarrow \psi[x/p+z])} \qquad \frac{\forall x \in \mathbb{Z}, \ldots \quad (p \geqslant x \wedge \varphi \Rightarrow \psi)}{\forall z \in \mathbb{N}, \ldots \quad (\varphi[x/p-z] \Rightarrow \psi[x/p-z])}$$

if x does not occur in the polynomial p

By Rule (A), the inequality constraint (16) is therefore transformed into

$$\forall y \in \mathbb{Z}, z \in \mathbb{N} \quad a_0 + a_1\,(y+z) + a_2\, y \geqslant c_0 \tag{17}$$

To replace all remaining quantifiers over $\mathbb{Z}$ by quantifiers over $\mathbb{N}$, we add the following rule. It splits the remaining inequality constraint φ (which may have additional universal quantifiers) into the cases where y is positive resp. negative.

B. **Split**

$$\frac{\forall y \in \mathbb{Z} \quad \varphi}{\forall y \in \mathbb{N} \quad \varphi \qquad \wedge \qquad \forall y \in \mathbb{N} \quad \varphi[y/-y]}$$

[18] For reasons of space, we do not present the remaining transformation rules of [11], which are applied in our implementation as well. These rules are used to delete "max" and to eliminate arbitrary conditions, e.g., conditions that are not removed by Rule (A). Similar transformation rules can for example also be found in [18].

Thus, Rule (B) transforms (17) into the conjunction of (18) and (19).

$$\forall y \in \mathbb{N}, z \in \mathbb{N} \quad a_0 + a_1\,(y+z) + a_2\,y \geqslant c_0 \tag{18}$$
$$\forall y \in \mathbb{N}, z \in \mathbb{N} \quad a_0 + a_1\,(-y+z) - a_2\,y \geqslant c_0 \tag{19}$$

If φ still has conditions, then a split by Rule (B) often results in unsatisfiable conditions. To detect them, we use SMT-solvers for linear integer arithmetic and additional sufficient criteria to detect also certain non-linear unsatisfiable conditions like $x^2 < 0$, etc. If a condition is found to be unsatisfiable, we delete the inequality constraint. Note that (18) can be reformulated as

$$\forall y \in \mathbb{N}, z \in \mathbb{N} \quad (a_1 + a_2)\,y + a_1\,z + (a_0 - c_0) \geqslant 0$$

So we now have to ensure non-negativeness of "polynomials" over variables like y and z that range over $\mathbb{N}$, where the "coefficients" are polynomials like "$a_1 + a_2$" over the abstract variables. To this end, it suffices to require that all these "coefficients" are $\geqslant 0$ [21]. In other words, now one can eliminate all universally quantified variables like y, z and transform (18) into the *Diophantine constraint*

$$a_1 + a_2 \geqslant 0 \qquad \wedge \qquad a_1 \geqslant 0 \qquad \wedge \qquad a_0 - c_0 \geqslant 0$$

C. **Eliminating Universally Quantified Variables**

$$\frac{\forall x_1 \in \mathbb{N}, \ldots, x_m \in \mathbb{N} \quad p_1\,x_1^{e_{11}} \ldots x_m^{e_{m1}} + \ldots + p_k\,x_1^{e_{1k}} \ldots x_m^{e_{mk}} \geqslant 0}{p_1 \geqslant 0 \;\wedge \ldots \wedge\; p_k \geqslant 0}$$

if the p_i do not contain variables from $\mathcal{V}$

To search for values of the abstract coefficients that satisfy the resulting Diophantine constraints, one fixes upper and lower bounds for these values. Then we showed in [10] how to translate such Diophantine constraints into a satisfiability problem for propositional logic which can be handled by SAT solvers efficiently. The constraints resulting from the initial inequality constraint (16) are for example satisfied by $a_0 = 0$, $a_1 = 1$, $a_2 = -1$, and $c_0 = 0$.[19] With these values, the abstract interpretation $a_0 + a_1\,x_1 + a_2\,x_2$ for SUM is turned into the concrete interpretation $x_1 - x_2$. With the resulting concrete I-interpretation $\mathcal{P}ol$, we would have $\mathcal{P}_{\succ} = \{(8)\}$ and $\mathcal{P}_{bound} = \{(7)\}$. The reduction pair processor of Thm. 9 would therefore transform the initial DP problem $\mathcal{P} = \{(7), (8)\}$ into the two problems $\mathcal{P} \setminus \mathcal{P}_{\succ} = \{(7)\}$ and $\mathcal{P} \setminus \mathcal{P}_{bound} = \{(8)\}$. Both of them are easy to solve (e.g., by using $\mathcal{P}ol'$ with $\mathsf{SUM}_{\mathcal{P}ol'} = 1$, $\mathsf{SIF}_{\mathcal{P}ol'} = 0$ and $\mathcal{P}ol''$ with $\mathsf{SUM}_{\mathcal{P}ol''} = 0$, $\mathsf{SIF}_{\mathcal{P}ol''} = 1$ or by using other processors like the *dependency graph*).

Our approach also directly works for ITRSs with extra variables on right-hand sides of rules. Then the rewrite relation is defined as $s \hookrightarrow_{\mathcal{R}} t$ iff there is a

[19] Note that the abstract coefficient c_0 can only occur in atomic Diophantine constraints of the form "$p - c_0 \geqslant 0$" where the polynomial p does not contain c_0. These constraints are always satisfied when choosing c_0 small enough. Therefore, one does not have to consider constraints with c_0 anymore and one also does not have to determine the actual value of c_0. This is advantageous for ITRSs with "large constants" like $\mathsf{f}(\mathsf{true}, x) \to \mathsf{f}(1000 \geqslant x, x+1)$, since current Diophantine constraint solvers like [10] usually only consider small ranges for the abstract coefficients.

rule $\ell \to r \in \mathcal{R} \cup \mathcal{PD}$ such that $s|_\pi = \ell\sigma$ and $t = s[r\sigma]_\pi$, $\ell\sigma$ does not contain redexes as proper subterms, and σ *is a normal substitution* (i.e., $\sigma(y)$ is in normal form also for variables y occurring only in r). Now we can also handle ITRSs with non-determinism like $\mathsf{f}(\mathsf{true}, x) \to \mathsf{f}(x > y \wedge x \geqslant \mathsf{0},\ y)$. Here, the argument x of f is replaced by an arbitrary smaller number y. Handling non-deterministic algorithms is often necessary for termination proofs of imperative programs when abstracting away "irrelevant" parts of the computation, cf. [4,8,24].

This also opens up a possibility to deal with algorithms that contain "large constants" computed by user-defined functions. For instance, consider an ITRS containing $\mathsf{f}(\mathsf{true},\ x) \ \to\ \mathsf{f}(\mathsf{ack}(\mathsf{10},\mathsf{10}) \geqslant x,\ x+1)$ and ack-rules computing the *Ackermann* function. With our approach, the ack-rules would have to be weakly decreasing, cf. Rule (V). This implies $\mathsf{ack}_{\mathcal{P}ol}(n, m) \geqslant \mathit{Ackermann}(n, m)$, which does not hold for any max-polynomial $\mathsf{ack}_{\mathcal{P}ol}$. But such examples can be handled by automatically transforming the original ITRS to an ITRS with extra variables whose termination implies termination of the original ITRS. If s is a ground term like $\mathsf{ack}(\mathsf{10},\mathsf{10})$ on the right-hand side of a rule and all usable rules of s are non-overlapping, then one can replace s by a fresh variable y. This variable must then be added as an additional argument. In this way, one can transform the f-rule above into the following ones. Termination of the new ITRS is easy to show.

$$\mathsf{f}(\mathsf{true}, x) \to \mathsf{f}'(\mathsf{true}, x, y) \qquad\qquad \mathsf{f}'(\mathsf{true}, x, y)\ \to\ \mathsf{f}'(y \geqslant x, x+1, y)$$

6 Experiments and Conclusion

We have adapted the DP framework in order to prove termination of ITRSs where integers are built-in. To evaluate our approach, we implemented it in our termination prover AProVE [15]. Of course, here we used appropriate strategies to control the application of the transformation rules from Sect. 4 and 5, since these are neither confluent nor equivalence-preserving. We tested our implementation on a data base of 117 ITRSs (including also *conditional* ITRSs, cf. Footnote 4). Our data base contains all 19 examples from the collection of [17] and all 29 examples from the collection[20] of [9] converted to integers, all 19 examples from the papers [2,3,4,5,6,7,8,24,25][21] on imperative programs converted to term rewriting, and several other "typical" algorithms on integers (including also some non-terminating ones). With a timeout of 1 minute for each example, the new version of AProVE with the contributions of this paper can prove termination of 104 examples (i.e., of 88.9 %). In particular, AProVE succeeds on all ITRSs mentioned in the current paper. In contrast, we also ran the previous version of AProVE (AProVE08) and the termination tool T$_2$ [22] that do not support built-in integers on this data base. Here, we converted integers into terms constructed with $\mathsf{0}$, s, pos, and neg and we added rules for the pre-defined operations on integers in this representation, cf. Sect. 1.[22] Although AProVE08 won the

[20] In these examples, (multi)sets were replaced by lists and those examples where the use of (multi)sets was essential were omitted.

[21] We omitted 4 examples from these papers that contain parallel processes or pointers.

[22] In this way, one can always convert DP problems for ITRSs to ordinary DP problems.

last *International Competition of Termination Provers* 2008 for term rewriting[23] and $\mathsf{T_TT_2}$ was second, both performed very poorly on the examples. AProVE08 could only prove termination of 24 of them (20.5 %) and $\mathsf{T_TT_2}$ proved termination of 6 examples (5.1 %). This clearly shows the enormous benefits of built-in integers in term rewriting. To access our collection of examples, for details on our experimental results, and to run the new version of AProVE via a web interface, we refer to `http://aprove.informatik.rwth-aachen.de/eval/Integer/`.

References

1. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoretical Computer Science 236, 133–178 (2000)
2. Bradley, A.R., Manna, Z., Sipma, H.B.: Termination of polynomial programs. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 113–129. Springer, Heidelberg (2005)
3. Bradley, A.R., Manna, Z., Sipma, H.B.: Linear ranking with reachability. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 491–504. Springer, Heidelberg (2005)
4. Chawdhary, A., Cook, B., Gulwani, S., Sagiv, M., Yang, H.: Ranking abstractions. In: Drossopoulou, S. (ed.) ESOP 2008. LNCS, vol. 4960, pp. 148–162. Springer, Heidelberg (2008)
5. Colón, M., Sipma, H.B.: Synthesis of linear ranking functions. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 67–81. Springer, Heidelberg (2001)
6. Colón, M., Sipma, H.B.: Practical methods for proving program termination. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 442–454. Springer, Heidelberg (2002)
7. Cook, B., Podelski, A., Rybalchenko, A.: Abstraction refinement for termination. In: Hankin, C., Siveroni, I. (eds.) SAS 2005. LNCS, vol. 3672, pp. 87–101. Springer, Heidelberg (2005)
8. Cook, B., Podelski, A., Rybalchenko, A.: Termination proofs for systems code. In: Proc. PLDI 2006, pp. 415–426. ACM Press, New York (2006)
9. Falke, S., Kapur, D.: Dependency pairs for rewriting with built-in numbers and semantic data structures. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 94–109. Springer, Heidelberg (2008)
10. Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: SAT solving for termination analysis with polynomial interpretations. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 340–354. Springer, Heidelberg (2007)
11. Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: Maximal termination. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 110–125. Springer, Heidelberg (2008)
12. Giesl, J., Arts, T., Ohlebusch, E.: Modular termination proofs for rewriting using dependency pairs. Journal of Symbolic Computation 34(1), 21–58 (2002)
13. Giesl, J., Thiemann, R., Schneider-Kamp, P.: The DP framework: Combining techniques for automated termination proofs. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS (LNAI), vol. 3452, pp. 301–331. Springer, Heidelberg (2005)

[23] For more information, see `http://termcomp.uibk.ac.at/`.

14. Giesl, J., Swiderski, S., Schneider-Kamp, P., Thiemann, R.: Automated termination analysis for Haskell: From term rewriting to programming languages. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 297–312. Springer, Heidelberg (2006)
15. Giesl, J., Schneider-Kamp, P., Thiemann, R.: AProVE 1.2: Automatic termination proofs in the DP framework. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS, vol. 4130, pp. 281–286. Springer, Heidelberg (2006)
16. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. Journal of Automated Reasoning 37(3), 155–203 (2006)
17. Giesl, J., Thiemann, R., Swiderski, S., Schneider-Kamp, P.: Proving termination by bounded increase. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 443–459. Springer, Heidelberg (2007)
18. Gulwani, S., Tiwari, A.: Constraint-based approach for analysis of hybrid systems. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 190–203. Springer, Heidelberg (2008)
19. Hirokawa, N., Middeldorp, A.: Automating the dependency pair method. Information and Computation 199(1,2), 172–199 (2005)
20. Hirokawa, N., Middeldorp, A.: Tyrolean Termination Tool: Techniques and features. Information and Computation 205(4), 474–511 (2007)
21. Hong, H., Jakuš, D.: Testing positiveness of polynomials. Journal of Automated Reasoning 21(1), 23–38 (1998)
22. Korp, M., Sternagel, C., Zankl, H., Middeldorp, A.: Tyrolean Termination Tool 2. In: Treinen, R. (ed.) RTA 2009. LNCS, vol. 5595, pp. 295–304. Springer, Heidelberg (2009)
23. Ohlebusch, E.: Termination of logic programs: Transformational approaches revisited. Appl. Algebra in Engineering, Comm. and Computing 12, 73–116 (2001)
24. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 239–251. Springer, Heidelberg (2004)
25. Podelski, A., Rybalchenko, A.: Transition invariants. In: LICS 2004, pp. 32–41 (2004)
26. Rubio, A.: Present and future of proving termination of rewriting. Invited talk, `www.risc.uni-linz.ac.at/about/conferences/rta2008/slides/Slides_Rubio.pdf`
27. Schneider-Kamp, P., Giesl, J., Serebrenik, A., Thiemann, R.: Automated termination proofs for logic programs by term rewriting. ACM Transactions on Computational Logic (to appear, 2009)
28. Serebrenik, A., De Schreye, D.: Inference of termination conditions for numerical loops in Prolog. Theory and Practice of Logic Programming 4(5,6), 719–751 (2004)

Dependency Pairs and Polynomial Path Orders*

Martin Avanzini and Georg Moser

Institute of Computer Science, University of Innsbruck, Austria
{martin.avanzini,georg.moser}@uibk.ac.at

Abstract. We show how polynomial path orders can be employed efficiently in conjunction with weak innermost dependency pairs to automatically certify polynomial runtime complexity of term rewrite systems and the polytime computability of the functions computed. The established techniques have been implemented and we provide ample experimental data to assess the new method.

1 Introduction

In order to measure the complexity of a (terminating) term rewrite system (TRS for short) it is natural to look at the maximal length of derivation sequences—the *derivation length*—as suggested by Hofbauer and Lautemann in [1]. More precisely, the *runtime complexity function* with respect to a (finite and terminating) TRS $\mathcal{R}$ relates the maximal derivation length to the size of the initial term, whenever the set of initial terms is restricted to constructor based terms, also called *basic* terms. The restriction to basic terms allows us to accurately express the runtime complexity of programs through the runtime complexity of TRSs. In this paper we study and combine recent efforts for the *automatic* analysis of runtime complexities of TRSs. In [2] we introduced a restriction of the multiset path order, called *polynomial path order* (POP* for short) that induces polynomial runtime complexity if restricted to innermost rewriting. The definition of POP* employs the idea of *tiered recursion* [3]. Syntactically this amounts to a separation of arguments into *normal* and *safe* arguments, cf. [4]. Furthermore, Hirokawa and the second author introduced a variant of dependency pairs, dubbed *weak dependency pairs*, that makes the dependency pair method applicable in the context of complexity analysis, cf. [5,6].

We show how weak innermost dependency pairs can be successfully applied in conjunction with POP*. The following example (see [7]) motivates this study. Consider the TRS $\mathcal{R}_{\mathsf{bits}}$ encoding the function $\lambda x.\lceil \log(x+1) \rceil$ for natural numbers given as tally sequences:

$$
\begin{array}{llll}
1\colon & \mathsf{half}(0) \to 0 & 4\colon & \mathsf{bits}(0) \to 0 \\
2\colon & \mathsf{half}(\mathsf{s}(0)) \to 0 & 5\colon & \mathsf{bits}(\mathsf{s}(0)) \to \mathsf{s}(0) \\
3\colon & \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{half}(x)) & 6\colon & \mathsf{bits}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{bits}(\mathsf{s}(\mathsf{half}(x))))
\end{array}
$$

* This research is partially supported by FWF (Austrian Science Fund) projects P20133.

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 48–62, 2009.
© Springer-Verlag Berlin Heidelberg 2009

It is easy to see that the TRS $\mathcal{R}_{\mathsf{bin}}$ is not compatible with POP*, even if we allow quasi-precedences, see Section 4. On the other hand, employing (weak innermost) dependency pairs, argument filtering, and the usable rules criteria in conjunction with POP*, polynomial innermost runtime complexity of $\mathcal{R}_{\mathsf{bin}}$ can be shown fully automatically.

The combination of dependency pairs and polynomial path orders turns out to be technically involved. One of the first obstacles one encounters is that the pair $(\gtrsim_{\mathsf{pop}*}, >_{\mathsf{pop}*})$ cannot be used as a reduction pair, as $\gtrsim_{\mathsf{pop}*}$ fails to be closed under contexts. (This holds a fortiori for *safe* reduction pairs, as studied in [5].) Conclusively, we start from scratch and study polynomial path orders in the context of *relative rewriting* [8]. Based on this study an incorporation of argument filterings becomes possible so that we can employ the pair $(\gtrsim^{\pi}_{\mathsf{pop}*}, >^{\pi}_{\mathsf{pop}*})$ in conjunction with dependency pairs successfully. Here, $>^{\pi}_{\mathsf{pop}*}$ refers to the order obtained by combining $>_{\mathsf{pop}*}$ with the argument filtering π as expected, and $\gtrsim^{\pi}_{\mathsf{pop}*}$ denotes the extension of $>^{\pi}_{\mathsf{pop}*}$ by term equivalence, preserving the separation of safe and normal argument positions. Note that for polynomial path orders, the integration of argument filterings is not only non-trivial, but indeed a challenging task. This is mainly due to the embodiment of tiered recursion in POP*. Thus we establish a combination of two syntactic techniques for automatic runtime complexity analysis. The experimental evidence given below indicates the power and in particular the efficiency of the provided results.

Our next contribution is concerned with *implicit complexity theory*, see for example [9]. A careful analysis of our main result shows that polynomial path orders in conjunction with (weak innermost) dependency pairs even induce polytime computability of the functions defined by the TRS studied. This result fits well with recent results by Marion and Péchoux on the use of restricted forms of the dependency pair method to characterise complexity classes like PTIME or PSPACE, cf. [10]. Both results allow to conclude, based on different restrictions, polytime computability of the functions defined by constructor TRSs, whose termination can be shown by the dependency pair method. Note that the results in [10] also capture programs that admit infeasible runtime complexities, but define functions that are computable in polytime, if suitable (and non-trivial) program transformations are used. Such programs are outside the scope of our results. Thus it seems that our results more directly assess the complexity of the given programs. Note that our tool provides (for the first time) a fully automatic application of the dependency pair method in the context of implicit complexity theory. Here we only want to mention that for a variant of the TRS $\mathcal{R}_{\mathsf{bin}}$, as studied in [10], our tool easily verifies polytime computability fully automatically.

The rest of the paper is organised as follows. In Section 2 we present basic notions and recall (briefly) the *path order for FP* from [11]. We then briefly recall dependency pairs in the context of complexity analysis from [5,6], cf. Section 3. In Section 4 we present polynomial path orders over quasi-precedences. Our main results are presented in Section 5. We continue with experimental results in Section 6, and conclude in Section 7.

2 The Polynomial Path Order on Sequences

We assume familiarity with the basics of term rewriting, see [12,13]. Let $\mathcal{V}$ denote a countably infinite set of variables and $\mathcal{F}$ a signature, containing at least one constant. The set of terms over $\mathcal{F}$ and $\mathcal{V}$ is denoted as $\mathcal{T}(\mathcal{F},\mathcal{V})$ and the set of ground terms as $\mathcal{T}(\mathcal{F})$. We write $\mathsf{Fun}(t)$ and $\mathsf{Var}(t)$ for the set of function symbols and variables appearing in t, respectively. The root symbol $\mathsf{rt}(t)$ of a term t is defined as usual and the (proper) subterm relation is denoted as $\trianglelefteq$ ($\triangleleft$). We write $s|_p$ for the *subterm* of s at position p. The *size* $|t|$ of a term t is defined as usual and the *width* of t is defined as $\mathsf{width}(t) := \max\{n, \mathsf{width}(t_1), \ldots, \mathsf{width}(t_n)\}$ if $t = f(t_1,\ldots,t_n)$ and $n > 0$ or $\mathsf{width}(t) = 1$ else. Let $\succsim$ be a preorder on the signature $\mathcal{F}$, called *quasi-precedence* or simply *precedence*. Based on $\succsim$ we define an equivalence $\approx$ on terms: $s \approx t$ if either (i) $s = t$ or (ii) $s = f(s_1,\ldots,s_n)$, $t = g(t_1,\ldots,t_n)$, $f \approx g$ and there exists a permutation π such that $s_i \approx t_{\pi(i)}$. For a preorder $\succsim$, we use $\succsim^{\mathsf{mul}}$ for the multiset extension of $\succsim$, which is again a preorder. The proper order (equivalence) induced by $\succsim^{\mathsf{mul}}$ is written as $\succ^{\mathsf{mul}}$ ($\approx^{\mathsf{mul}}$).

A *term rewrite system* (*TRS* for short) $\mathcal{R}$ over $\mathcal{T}(\mathcal{F},\mathcal{V})$ is a *finite* set of rewrite rules $l \to r$, such that $l \notin \mathcal{V}$ and $\mathsf{Var}(l) \supseteq \mathsf{Var}(r)$. We write $\to_{\mathcal{R}}$ ($\xrightarrow{\mathsf{i}}_{\mathcal{R}}$) for the induced (innermost) rewrite relation. The set of defined function symbols is denoted as $\mathcal{D}$, while the constructor symbols are collected in $\mathcal{C}$, clearly $\mathcal{F} = \mathcal{D} \cup \mathcal{C}$. We use $\mathsf{NF}(\mathcal{R})$ to denote the set of normal forms of $\mathcal{R}$ and set $\mathsf{Val} := \mathcal{T}(\mathcal{C},\mathcal{V})$. The elements of Val are called *values*. A TRS is called *completely defined* if normal forms coincide with values. We define $\mathcal{T}_{\mathsf{b}} := \{f(v_1,\ldots,v_n) \mid f \in \mathcal{D} \text{ and } v_i \in \mathsf{Val}\}$ as the set of *basic terms*. A TRS $\mathcal{R}$ is a *constructor TRS* if $l \in \mathcal{T}_{\mathsf{b}}$ for all $l \to r \in \mathcal{R}$. Let $\mathcal{Q}$ denote a TRS. The *generalised restricted rewrite relation* $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ is the restriction of $\to_{\mathcal{R}}$ where all arguments of the redex are in normal form with respect to the TRS $\mathcal{Q}$ (compare [14]). We define the (innermost) relative rewriting relation (denoted as $\xrightarrow{\mathsf{i}}_{\mathcal{R}/\mathcal{S}}$) as follows:

$$\xrightarrow{\mathsf{i}}_{\mathcal{R}/\mathcal{S}} := \xrightarrow{\mathcal{R}\cup\mathcal{S}}{}^{*}_{\mathcal{S}} \cdot \xrightarrow{\mathcal{R}\cup\mathcal{S}}_{\mathcal{R}} \cdot \xrightarrow{\mathcal{R}\cup\mathcal{S}}{}^{*}_{\mathcal{S}} ,$$

Similarly, we set $\xrightarrow{\mathsf{i}}{}^{\varepsilon}_{\mathcal{R}/\mathcal{S}} := \xrightarrow{\mathcal{R}\cup\mathcal{S}}{}^{*}_{\mathcal{S}} \cdot \xrightarrow{\mathcal{R}\cup\mathcal{S}}{}^{\varepsilon}_{\mathcal{R}} \cdot \xrightarrow{\mathcal{R}\cup\mathcal{S}}{}^{*}_{\mathcal{S}}$, to define an *(innermost) relative root-step*.

A *polynomial interpretation* is a well-founded and monotone algebra $(\mathcal{A},>)$ with carrier $\mathbb{N}$ such that $>$ is the usual order on natural numbers and all interpretation functions $f_{\mathcal{A}}$ are polynomials. Let $\alpha\colon \mathcal{V} \to \mathcal{A}$ denote an *assignment*, then we write $[\alpha]_{\mathcal{A}}(t)$ for the evaluation of term t with respect to $\mathcal{A}$ and α. A polynomial interpretation is called a *strongly linear interpretation* (*SLI* for short) if all function symbols are interpreted by *weight functions* $f_{\mathcal{A}}(x_1,\ldots,x_n) = \sum_{i=1}^{n} x_i + c$ with $c \in \mathbb{N}$. The *derivation length* of a terminating term s with respect to $\to$ is defined as $\mathrm{dl}(s,\to) := \max\{n \mid \exists t.\ s \to^n t\}$, where $\to^n$ denotes the n-fold application of $\to$. The *innermost runtime complexity function* $\mathrm{rc}^{\mathsf{i}}_{\mathcal{R}}$ with respect to a TRS $\mathcal{R}$ is defined as $\mathrm{rc}^{\mathsf{i}}_{\mathcal{R}}(n) := \max\{\mathrm{dl}(t,\xrightarrow{\mathsf{i}}_{\mathcal{R}}) \mid t \in \mathcal{T}_{\mathsf{b}} \text{ and } |t| \leqslant n\}$. If no confusion can arise $\mathrm{rc}^{\mathsf{i}}_{\mathcal{R}}$ is simply called *runtime complexity function*.

We recall the bare essentials of the polynomial path order $\blacktriangleright$ on sequences (POP for short) as put forward in [11]. We kindly refer the reader to [11,2] for

motivation and examples. We recall the definition of *finite approximations* $\blacktriangleright_k^l$ of $\blacktriangleright$. The latter is conceived as the *limit* of these approximations. The domain of this order are so-called *sequences* $\mathcal{S}\mathrm{eq}(\mathcal{F},\mathcal{V}) := \mathcal{T}(\mathcal{F}\cup\{\circ\},\mathcal{V})$. Here $\mathcal{F}$ is a finite signature and $\circ \notin \mathcal{F}$ a fresh variadic function symbol, used to form sequences. We denote sequences $\circ(s_1,\dots,s_n)$ by $[s_1\cdots s_n]$ and write $a :: [b_1\cdots b_n]$ for the sequence $[a\ b_1\cdots b_n]$.

Let $\succsim$ denote a precedence. The order $\blacktriangleright_k^l$ is based on an auxiliary order $\trianglerighteq_k^l$. Below we set $\gtrsim_k^l := \trianglerighteq_k^l \cup \approx$, where $\approx$ denotes the equivalence on terms defined above. We write $\{\!\{t_1,\dots,t_n\}\!\}$ to denote multisets and $\uplus$ for the multiset sum.

Definition 1. *Let $k,l \geqslant 1$. The order $\trianglerighteq_k^l$ induced by $\succsim$ is inductively defined as follows: $s \trianglerighteq_k^l t$ for $s = f(s_1,\dots,s_n)$ or $s = [s_1\cdots s_n]$ if either*

(1) $s_i \gtrsim_k^l t$ for some $i \in \{1,\dots,n\}$, or
(2) $s = f(s_1,\dots,s_n)$, $t = g(t_1,\dots,t_m)$ with $f \succ g$ or $t = [t_1\cdots t_m]$, $s \trianglerighteq_k^{l-1} t_j$ for all $j \in \{1,\dots,m\}$, and $m < k + \mathsf{width}(s)$,
(3) $s = [s_1\cdots s_n]$, $t = [t_1\cdots t_m]$ and the following properties hold:
- *$\{\!\{t_1,\dots,t_m\}\!\} = N_1 \uplus \cdots \uplus N_n$ for some multisets $N_1,\dots,N_n$, and*
- *there exists $i \in \{1,\dots,n\}$ such that $\{\!\{s_i\}\!\} \not\approx^{\mathsf{mul}} N_i$, and*
- *for all $1 \leqslant i \leqslant n$ such that $\{\!\{s_i\}\!\} \not\approx^{\mathsf{mul}} N_i$ we have $s_i \trianglerighteq_k^l r$ for all $r \in N_i$, and $m < k + \mathsf{width}(s)$.*

Definition 2. *Let $k,l \geqslant 1$. The* approximation $\blacktriangleright_k^l$ *of the polynomial path order on sequences induced by $\succsim$ is inductively defined as follows: $s \blacktriangleright_k^l t$ for $s = f(s_1,\dots,s_n)$ or $s = [s_1\cdots s_n]$ if either $s \trianglerighteq_k^l t$ or*

(1) $s_i \blacktriangleright_k^l t$ for some $i \in \{1,\dots,n\}$,
(2) $s = f(s_1,\dots,s_n)$, $t = [t_1\cdots t_m]$, and the following properties hold:
- *$s \blacktriangleright_k^{l-1} t_{j_0}$ for some $j_0 \in \{1,\dots,m\}$,*
- *$s \trianglerighteq_k^{l-1} t_j$ for all $j \neq j_0$, and $m < k + \mathsf{width}(s)$,*

(3) $s = f(s_1,\dots,s_n)$, $t = g(t_1,\dots,t_m)$, $f \sim g$ and $[s_1\cdots s_n] \blacktriangleright_k^l [t_1\cdots t_m]$, or
(4) $s = [s_1\cdots s_n]$, $t = [t_1\cdots t_m]$ and the following properties hold:
- *$\{\!\{t_1,\dots,t_m\}\!\} = N_1 \uplus \cdots \uplus N_n$ for some multisets $N_1,\dots,N_n$, and*
- *there exists $i \in \{1,\dots,n\}$ such that $\{\!\{s_i\}\!\} \not\approx^{\mathsf{mul}} N_i$, and*
- *for all $1 \leqslant i \leqslant n$ such that $\{\!\{s_i\}\!\} \not\approx^{\mathsf{mul}} N_i$ we have $s_i \blacktriangleright_k^l r$ for all $r \in N_i$, and $m < k + \mathsf{width}(s)$.*

Above we set $\blacktriangleright_k^l := \blacktriangleright_k^l \cup \approx$ and abbreviate $\blacktriangleright_k^k$ as $\blacktriangleright_k$ in the following. Note that the empty sequence $[\,]$ is minimal with respect to both orders.

It is easy to see that for $k \leqslant l$, we have $\trianglerighteq_k \subseteq \trianglerighteq_l$ and $\blacktriangleright_k \subseteq \blacktriangleright_l$ and that $s \blacktriangleright_k t$ implies that $\mathsf{width}(t) < \mathsf{width}(s) + k$. For a fixed approximation $\blacktriangleright_k$, we define the length of its longest decent as: $\mathsf{G}_k(t) := \max\{n \mid t = t_0 \blacktriangleright_k t_1 \blacktriangleright_k \cdots \blacktriangleright_k t_n\}$. The following proposition is a reformulation of Lemma 6 in [11].

Proposition 3. *Let $k \in \mathbb{N}$. There exists a polynomial interpretation $\mathcal{A}$ such that $\mathsf{G}_k(t) \leqslant [\alpha]_{\mathcal{A}}(t)$ for all assignments $\alpha : \mathcal{V} \to \mathbb{N}$. As a consequence, for all terms $f(t_1,\dots,t_n)$ with $[\alpha]_{\mathcal{A}}(t_i) = \mathsf{O}(|t_i|)$, $\mathsf{G}_k(f(t_1,\dots,t_n))$ is bounded by a polynomial p in the size of t, where p depends on k only.*

Note that the polynomial interpretation $\mathcal{A}$ employed in the proposition fulfils: $\circ_{\mathcal{A}}(m_1,\dots,m_n) = \sum_{i=1}^n m_i + n$. In particular, we have $[\alpha]_{\mathcal{A}}([\,]) = 0$.

3 Complexity Analysis Based on the Dependency Pair Method

In this section, we briefly recall the central definitions and results established in [5,6]. We kindly refer the reader to [5,6] for further examples and underlying intuitions. Let $\mathcal{X}$ be a set of symbols. We write $C\langle t_1, \ldots, t_n\rangle_{\mathcal{X}}$ to denote $C[t_1, \ldots, t_n]$, whenever $\mathsf{rt}(t_i) \in \mathcal{X}$ for all $i \in \{1, \ldots, n\}$ and C is a n-hole context containing no symbols from $\mathcal{X}$. We set $\mathcal{D}^\sharp := \mathcal{D} \cup \{f^\sharp \mid f \in \mathcal{D}\}$ with each $f^\sharp$ a fresh function symbol. Further, for $t = f(t_1, \ldots, t_n)$ with $f \in \mathcal{D}$, we set $t^\sharp := f^\sharp(t_1, \ldots, t_n)$.

Definition 4. *Let $\mathcal{R}$ be a TRS. If $l \to r \in \mathcal{R}$ and $r = C\langle u_1, \ldots, u_n\rangle_{\mathcal{D}}$ then $l^\sharp \to \mathrm{COM}(u_1^\sharp, \ldots, u_n^\sharp)$ is called a* weak innermost dependency pair *of $\mathcal{R}$. Here $\mathrm{COM}(t) = t$ and $\mathrm{COM}(t_1, \ldots, t_n) = \mathsf{c}(t_1, \ldots, t_n)$, $n \neq 1$, for a fresh constructor symbol c, the* compound symbol. *The set of all weak innermost dependency pairs is denoted by $\mathsf{WIDP}(\mathcal{R})$.*

Example 5. Reconsider the example $\mathcal{R}_{\mathsf{bits}}$ from the introduction. The set of weak innermost dependency pairs $\mathsf{WIDP}(\mathcal{R}_{\mathsf{bits}})$ is given by

$$
\begin{array}{llll}
7\colon & \mathsf{half}^\sharp(0) \to \mathsf{c}_1 & 10\colon & \mathsf{bits}^\sharp(0) \to \mathsf{c}_3 \\
8\colon & \mathsf{half}^\sharp(\mathsf{s}(0)) \to \mathsf{c}_2 & 11\colon & \mathsf{bits}^\sharp(\mathsf{s}(0)) \to \mathsf{c}_4 \\
9\colon & \mathsf{half}^\sharp(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{half}^\sharp(x) & 12\colon & \mathsf{bits}^\sharp(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{bits}^\sharp(\mathsf{s}(\mathsf{half}(x)))
\end{array}
$$

We write $f \rhd_{\mathrm{d}} g$ if there exists a rewrite rule $l \to r \in \mathcal{R}$ such that $f = \mathsf{rt}(l)$ and g is a defined symbol in $\mathsf{Fun}(r)$. For a set $\mathcal{G}$ of defined symbols we denote by $\mathcal{R}{\restriction}\mathcal{G}$ the set of rewrite rules $l \to r \in \mathcal{R}$ with $\mathsf{rt}(l) \in \mathcal{G}$. The set $\mathcal{U}(t)$ of usable rules of a term t is defined as $\mathcal{R}{\restriction}\{g \mid f \rhd_{\mathrm{d}}^* g \text{ for some } f \in \mathsf{Fun}(t)\}$. Finally, we define $\mathcal{U}(\mathcal{P}) = \bigcup_{l \to r \in \mathcal{P}} \mathcal{U}(r)$.

Example 6 (Example 5 continued). The usable rules of $\mathsf{WIDP}(\mathcal{R}_{\mathsf{bits}})$ consist of the following rules: $1\colon \mathsf{half}(0) \to 0$, $2\colon \mathsf{half}(\mathsf{s}(0)) \to 0$, and $3\colon \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{half}(x)$.

The following proposition allows the analysis of the (innermost) runtime complexity through the study of (innermost) relative rewriting, see [5] for the proof.

Proposition 7. *Let $\mathcal{R}$ be a TRS, let t be a basic terminating term, and let $\mathcal{P} = \mathsf{WIDP}(\mathcal{R})$. Then $\mathrm{dl}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}}) \leqslant \mathrm{dl}(t^\sharp, \xrightarrow{\mathsf{i}}_{\mathcal{U}(\mathcal{P}) \cup \mathcal{P}})$. Moreover, suppose $\mathcal{P}$ is non-duplicating and $\mathcal{U}(\mathcal{P}) \subseteq >_{\mathcal{A}}$ for some SLI $\mathcal{A}$. Then there exist constants $K, L \geqslant 0$ (depending on $\mathcal{P}$ and $\mathcal{A}$ only) such that $\mathrm{dl}(t, \xrightarrow{\mathsf{i}}_{\mathcal{R}}) \leqslant K \cdot \mathrm{dl}(t^\sharp, \xrightarrow{\mathsf{i}}_{\mathcal{P}/\mathcal{U}(\mathcal{P})}) + L \cdot |t^\sharp|$.*

This approach admits also an integration of *dependency graphs* [15] in the context of complexity analysis. The nodes of the *weak innermost dependency graph* $\mathsf{WIDG}(\mathcal{R})$ are the elements of $\mathcal{P}$ and there is an arrow from $s \to t$ to $u \to v$ if there exist a context C and substitutions σ, τ such that $t\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}}^* C[u\tau]$. Let $\mathcal{G} = \mathsf{WIDG}(\mathcal{R})$; a *strongly connected component* (*SCC* for short) in $\mathcal{G}$ is a maximal *strongly connected subgraph*. We write $\mathcal{G}/_{\equiv}$ for the *congruence graph*, where $\equiv$ is the equivalence relation induced by SCCs.

Example 8 (Example 5 continued). $\mathcal{G} = \mathsf{WIDG}(\mathcal{R}_{\mathsf{bits}})$ consists of the nodes (7)–(12) as mentioned in Example 5 and has the following shape:

$$7 \longleftarrow 9 \longrightarrow 8 \qquad 10 \qquad 12 \longrightarrow 11$$

The only non-trivial SCCs in $\mathcal{G}$ are $\{9\}$ and $\{12\}$. Hence $\mathcal{G}/_{\equiv}$ consists of the nodes $[7]_{\equiv}$–$[12]_{\equiv}$, and edges $([a]_{\equiv}, [b]_{\equiv})$ for edges (a, b) in $\mathcal{G}$. Here $[a]_{\equiv}$ denotes the equivalence class of a.

We set $\mathsf{L}(t) := \max\{\mathrm{dl}(t, \xrightarrow{\mathsf{i}}_{\mathcal{P}_m/\mathcal{S}}) \mid (\mathcal{P}_1, \ldots, \mathcal{P}_m) \text{ a path in } \mathcal{G}/_{\equiv}, \mathcal{P}_1 \in \mathsf{Src}\}$, where Src denotes the set of source nodes from $\mathcal{G}/_{\equiv}$ and $\mathcal{S} = \mathcal{P}_1 \cup \cdots \cup \mathcal{P}_{m-1} \cup \mathcal{U}(\mathcal{P}_1 \cup \cdots \cup \mathcal{P}_m)$. The proposition allows the use of different techniques to analyse polynomial runtime complexity on separate paths, cf. [6].

Proposition 9. *Let $\mathcal{R}$, $\mathcal{P}$, and t be as above. Then there exists a polynomial p (depending only on $\mathcal{R}$) such that* $\mathrm{dl}(t^\sharp, \xrightarrow{\mathsf{i}}_{\mathcal{P}/\mathcal{U}(\mathcal{P})}) \leqslant p(\mathsf{L}(t^\sharp))$.

4 The Polynomial Path Order over Quasi-precedences

In this section, we briefly recall the central definitions and results established in [2,16] on the *polynomial path order*. We employ the variant of POP* based on quasi-precendences, cf. [16].

As mentioned in the introduction, POP* relies on tiered recursion, which is captured by the notion of *safe mapping*. A *safe mapping* safe is a function that associates with every n-ary function symbol f the set of *safe argument positions*. If $f \in \mathcal{D}$ then $\mathsf{safe}(f) \subseteq \{1, \ldots, n\}$, for $f \in \mathcal{C}$ we fix $\mathsf{safe}(f) = \{1, \ldots, n\}$. The argument positions not included in $\mathsf{safe}(f)$ are called *normal* and denoted by $\mathsf{nrm}(f)$. We extend safe to terms $t \notin \mathcal{V}$. We define $\mathsf{safe}(f(t_1, \ldots, t_n)) := \{t_{i_1}, \ldots, t_{i_p}\}$ where $\mathsf{safe}(f) = \{i_1, \ldots, i_p\}$, and we define $\mathsf{nrm}(f(t_1, \ldots, t_n)) := \{t_{j_1}, \ldots, t_{j_q}\}$ where $\mathsf{nrm}(f) = \{j_1, \ldots, j_q\}$. Not every precedence is suitable for $>_{\mathsf{pop}*}$, in particular we need to assert that constructors are minimal.

We say that a precedence $\succsim$ is *admissible* for POP* if the following is satisfied: (i) $f \succ g$ with $g \in \mathcal{D}$ implies $f \in \mathcal{D}$, and (ii) if $f \approx g$ then $f \in \mathcal{D}$ if and only if $g \in \mathcal{D}$. In the sequel we assume any precedence is admissible. We extend the equivalence $\approx$ to the context of safe mappings: $s \overset{\mathsf{safe}}{\approx} t$, if (i) $s = t$, or (ii) $s = f(s_1, \ldots, s_n)$, $t = g(t_1, \ldots, t_n)$, $f \approx g$ and there exists a permutation π so that $s_i \overset{\mathsf{safe}}{\approx} t_{\pi(i)}$, where $i \in \mathsf{safe}(f)$ if and only if $\pi(i) \in \mathsf{safe}(g)$ for all $i \in \{1, \ldots, n\}$. Similar to POP, the definition of the polynomial path order $>_{\mathsf{pop}*}$ makes use of an auxiliary order $>_{\mathsf{pop}}$.

Definition 10. *The auxiliary order $>_{\mathsf{pop}}$ induced by $\succsim$ and safe is inductively defined as follows: $s = f(s_1, \ldots, s_n) >_{\mathsf{pop}} t$ if either*

(1) $s_i \gtrsim_{\mathsf{pop}} t$ for some $i \in \{1, \ldots, n\}$, and if $f \in \mathcal{D}$ then $i \in \mathsf{nrm}(f)$, or

(2) $t = g(t_1, \ldots, t_m)$, $f \succ g$, $f \in \mathcal{D}$ and $s >_{\mathsf{pop}} t_j$ for all $j \in \{1, \ldots, m\}$.

Definition 11. *The* polynomial path order $>_{\mathsf{pop}*}$ *induced by* $\succsim$ *and* safe *is inductively defined as follows:* $s = f(s_1, \dots, s_n) >_{\mathsf{pop}*} t$ *if either* $s >_{\mathsf{pop}} t$ *or*

(1) $s_i \gtrsim_{\mathsf{pop}*} t$ *for some* $i \in \{1, \dots, n\}$*, or*
(2) $t = g(t_1, \dots, t_m)$*,* $f \succ g$*,* $f \in \mathcal{D}$*, and*
  – $s >_{\mathsf{pop}*} t_{j_0}$ *for some* $j_0 \in \mathsf{safe}(g)$*, and*
  – *for all* $j \neq j_0$ *either* $s >_{\mathsf{pop}} t_j$*, or* $s \rhd t_j$ *and* $j \in \mathsf{safe}(g)$*, or*
(3) $t = g(t_1, \dots, t_m)$*,* $f \approx g$*,* $\mathsf{nrm}(s) >^{\mathsf{mul}}_{\mathsf{pop}*} \mathsf{nrm}(t)$ *and* $\mathsf{safe}(s) \gtrsim^{\mathsf{mul}}_{\mathsf{pop}*} \mathsf{safe}(t)$*.*

Above we set $\gtrsim_{\mathsf{pop}} := >_{\mathsf{pop}} \cup \overset{\mathsf{safe}}{\approx}$ and $\gtrsim_{\mathsf{pop}*} := >_{\mathsf{pop}*} \cup \overset{\mathsf{safe}}{\approx}$. Here $>^{\mathsf{mul}}_{\mathsf{pop}*}$ and $\gtrsim^{\mathsf{mul}}_{\mathsf{pop}*}$ refer to the strict and weak multiset extension of $\gtrsim_{\mathsf{pop}*}$ respectively.

The intuition of $>_{\mathsf{pop}}$ is to deny any recursive call, whereas $>_{\mathsf{pop}*}$ allows predicative recursion: by the restrictions imposed by the mapping safe, recursion needs to be performed on normal arguments, while a recursively computed result must only be used in a safe argument position, compare [4]. Note that the alternative $s \rhd t_j$ for $j \in \mathsf{safe}(g)$ in Definition 11(2) guarantees that POP* characterises the class of polytime computable functions. The proof of the next theorem follows the pattern of the proof of the Main Theorem in [2], but the result is stronger due to the extension to quasi-precedences.

Theorem 12. *Let* $\mathcal{R}$ *be a constructor TRS. If* $\mathcal{R}$ *is compatible with* $>_{\mathsf{pop}*}$*, i.e.,* $\mathcal{R} \subseteq >_{\mathsf{pop}*}$*, then the innermost runtime complexity* $\mathrm{rc}^{\mathsf{i}}_{\mathcal{R}}$ *induced is polynomially bounded.*

Note that Theorem 12 is too weak to handle the TRS $\mathcal{R}_{\mathsf{bits}}$ as the (necessary) restriction to an admissible precedence is too strong. To rectify this, we analyse POP* in Section 5 in the context of relative rewriting.

An argument filtering (for a signature $\mathcal{F}$) is a mapping π that assigns to every n-ary function symbol f an argument position $i \in \{1, \dots, n\}$ or a (possibly empty) list $\{k_1, \dots, k_m\}$ of argument positions with $1 \leqslant k_1 < \cdots < k_m \leqslant n$. The signature $\mathcal{F}_\pi$ consists of all function symbols f such that $\pi(f)$ is some list $\{k_1, \dots, k_m\}$, where in $\mathcal{F}_\pi$ the arity of f is m. Every argument filtering π induces a mapping from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}_\pi, \mathcal{V})$, also denoted by π:

$$\pi(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ \pi(t_i) & \text{if } t = f(t_1, \dots, t_n) \text{ and } \pi(f) = i \\ f(\pi(t_{k_1}), \dots, \pi(t_{k_m})) & \text{if } t = f(t_1, \dots, t_n) \text{ and } \pi(f) = \{k_1, \dots, k_m\} \end{cases}.$$

Definition 13. *Let* π *denote an argument filtering, and* $>_{\mathsf{pop}*}$ *a polynomial path order. We define* $s >^{\pi}_{\mathsf{pop}*} t$ *if and only if* $\pi(s) >_{\mathsf{pop}*} \pi(t)$*, and likewise* $s \gtrsim^{\pi}_{\mathsf{pop}*} t$ *if and only if* $\pi(s) \gtrsim_{\mathsf{pop}*} \pi(t)$*.*

Example 14 (Example 5 continued). Let π be defined as follows: $\pi(\mathsf{half}) = 1$ and $\pi(f) = \{1, \dots, n\}$ for each n-ary function symbol other than half. Compatibility of $\mathsf{WIDP}(\mathcal{R}_{\mathsf{bits}})$ with $>^{\pi}_{\mathsf{pop}*}$ amounts to the following set of order constraints:

$$\mathsf{half}^\sharp(0) >_{\mathsf{pop}*} \mathsf{c}_1 \quad \mathsf{bits}^\sharp(0) >_{\mathsf{pop}*} \mathsf{c}_3 \quad \mathsf{half}^\sharp(\mathsf{s}(\mathsf{s}(x))) >_{\mathsf{pop}*} \mathsf{half}^\sharp(x)$$
$$\mathsf{half}^\sharp(\mathsf{s}(0)) >_{\mathsf{pop}*} \mathsf{c}_2 \quad \mathsf{bits}^\sharp(\mathsf{s}(0)) >_{\mathsf{pop}*} \mathsf{c}_4 \quad \mathsf{bits}^\sharp(\mathsf{s}(\mathsf{s}(x))) >_{\mathsf{pop}*} \mathsf{bits}^\sharp(\mathsf{s}(x))$$

In order to define a POP* instance $>_{\mathsf{pop}*}$, we set $\mathsf{safe}(\mathsf{bits}^\sharp) = \mathsf{safe}(\mathsf{half}) = \mathsf{safe}(\mathsf{half}^\sharp) = \varnothing$ and $\mathsf{safe}(\mathsf{s}) = \{1\}$. Furthermore, we define an (admissible) precedence: $0 \approx \mathsf{c}_1 \approx \mathsf{c}_2 \approx \mathsf{c}_3 \approx \mathsf{c}_4$. The easy verification of $\mathsf{WIDP}(\mathcal{R}_{\mathsf{bits}}) \subseteq >^\pi_{\mathsf{pop}*}$ is left to the reader.

5 Dependency Pairs and Polynomial Path Orders

Motivated by Example 14, we show in this section that the pair $(\gtrsim^\pi_{\mathsf{pop}*}, >^\pi_{\mathsf{pop}*})$ can play the role of a *safe* reduction pair, cf. [5,6]. Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$ that is innermost terminating. In the sequel $\mathcal{R}$ is kept fixed. Moreover, we fix some safe mapping safe, an admissible precedence $\succsim$, and an argument filtering π. We refer to the induced POP* instance by $>^\pi_{\mathsf{pop}*}$.

We adapt safe to $\mathcal{F}_\pi$ in the obvious way: for each $f_\pi \in \mathcal{F}_\pi$ with corresponding $f \in \mathcal{F}$, we define $\mathsf{safe}(f_\pi) := \mathsf{safe}(f) \cap \pi(f)$, and likewise $\mathsf{nrm}(f_\pi) := \mathsf{nrm}(f) \cap \pi(f)$. Set $\mathsf{Val}_\pi := \mathcal{T}(\mathcal{C}_\pi, \mathcal{V})$. Based on $\mathcal{F}_\pi$ we define the *normalised signature* $\mathcal{F}^\mathsf{n}_\pi := \{f^\mathsf{n} \mid f \in \mathcal{F}_\pi\}$ where the arity of f^n is $|\mathsf{nrm}(f)|$. We extend $\succsim$ to $\mathcal{F}^\mathsf{n}_\pi$ by $f^\mathsf{n} \succsim g^\mathsf{n}$ if and only if $f \succsim g$. Let s be a fresh constant that is minimal with respect to $\succsim$. We introduce the *Buchholz norm* of t (denoted as $\|t\|$) as a term complexity measure that fits well with the definition of POP*. Set $\|t\| := 1 + \max\{n, \|t_1\|, \ldots, \|t_n\|\}$ for $t = f(t_1, \ldots, t_n)$ and $\|t\| := 1$, otherwise.

In the following we define an embedding from the relative rewriting relation $\xrightarrow{\mathsf{i}}{}^\varepsilon_{\mathcal{R}/\mathcal{S}}$ into $\blacktriangleright_k$, such that k depends only on TRSs $\mathcal{R}$ and $\mathcal{S}$. This embedding provides the technical tool to measure the number of root steps in a given derivation through the number of descents in $\blacktriangleright_k$. Hence Proposition 3 becomes applicable to establishing our main result. This intuition is cast into the next definition.

Definition 15. *A* predicative interpretation *is a pair of mappings* $(\mathsf{S}_\pi, \mathsf{N}_\pi)$ *from terms to sequences* $\mathcal{S}\mathrm{eq}(\mathcal{F}^\mathsf{n}_\pi \cup \{\mathsf{s}\}, \mathcal{V})$ *defined as follows. We assume* $\pi(t) = f(\pi(t_1), \ldots, \pi(t_n))$, $\mathsf{safe}(f) = \{i_1, \ldots, i_p\}$, *and* $\mathsf{nrm}(f) = \{j_1, \ldots, j_q\}$.

$$\mathsf{S}_\pi(t) := \begin{cases} [\,] & \text{if } \pi(t) \in \mathsf{Val}_\pi, \\ [f^\mathsf{n}(\mathsf{N}_\pi(t_{j_1}), \ldots, \mathsf{N}_\pi(t_{j_q}))\ \mathsf{S}_\pi(t_{i_1})\ \cdots\ \mathsf{S}_\pi(t_{i_p})] & \text{if } \pi(t) \notin \mathsf{Val}_\pi. \end{cases}$$

$$\mathsf{N}_\pi(t) := \mathsf{S}_\pi(t) :: \mathsf{BN}_\pi(t)$$

Here the function BN_π *maps a term* t *to the sequence* $[\mathsf{s} \cdots \mathsf{s}]$ *with* $\|\pi(t)\|$ *occurrences of the constant* s.

As a direct consequence of the definitions we have $\mathsf{width}(\mathsf{N}_\pi(t)) = \|\pi(t)\| + 1$ for all terms t.

Lemma 16. *There exists a polynomial* p *such that* $\mathsf{G}_k(\mathsf{N}_\pi(t)) \leqslant p(|t|)$ *for every basic term* t. *The polynomial* p *depends only on* k.

Proof. Suppose $t = f(v_1, \ldots, v_n)$ is a basic term with $\mathsf{safe}(f) = \{i_1, \ldots, i_p\}$ and $\mathsf{nrm}(f) = \{j_1, \ldots, j_q\}$. The only non-trivial case is when $\pi(t) \notin \mathsf{Val}_\pi$. Then $\mathsf{N}_\pi(t) = [u\ \mathsf{S}_\pi(v_{i_1}) \cdots \mathsf{S}_\pi(v_{i_p})] :: \mathsf{BN}_\pi(t)$ where $u = f^\mathsf{n}(\mathsf{N}_\pi(v_{j_1}), \ldots, \mathsf{N}_\pi(v_{j_q}))$.

Note that $\mathsf{S}_\pi(v_i) = [\,]$ for $i \in \{i_1, \dots, i_q\}$. Let $\mathcal{A}$ denote a polynomial interpretation fulfilling Proposition 3. Using the assumption $\circ_{\mathcal{A}}(m_1, \dots, m_n) = \sum_{i=1}^{n} m_i + n$, it is easy to see that $\mathsf{G}_k(\mathsf{N}_\pi(t))$ is bounded linear in $\|t\| \leqslant |t|$ and $[\alpha]_{\mathcal{A}}(u)$. As $\mathsf{N}_\pi(v_j) = [[]\ \mathsf{s} \cdots \mathsf{s}]$ with $\|\pi(v_j)\| \leqslant |t|$ occurrences of s, $\mathsf{G}_k(\mathsf{N}_\pi(v_j))$ is linear in $|t|$. Hence from Proposition 3 we conclude that $\mathsf{G}_k(\mathsf{N}_\pi(t))$ is polynomially bounded in $|t|$. □

The next sequence of lemmas shows that the relative rewriting relation $\xrightarrow{\mathsf{i},\varepsilon}_{\mathcal{R}/\mathcal{S}}$ is embeddable into $\blacktriangleright_k$.

Lemma 17. *Suppose* $s >^{\pi}_{\mathsf{pop}*} t$ *such that* $\pi(s\sigma) \in \mathsf{Val}_\pi$. *Then* $\mathsf{S}_\pi(s\sigma) = [\,] = \mathsf{S}_\pi(t\sigma)$ *and* $\mathsf{N}_\pi(s\sigma) \blacktriangleright_1 \mathsf{N}_\pi(t\sigma)$.

Proof. Let $\pi(s\sigma) \in \mathsf{Val}_\pi$, and suppose $s >^{\pi}_{\mathsf{pop}*} t$, i.e., $\pi(s) >_{\mathsf{pop}*} \pi(t)$ holds. Since $\pi(s) \in \mathsf{Val}_\pi$ (and due to our assumptions on safe mappings) only clause (1) from the definition of $>_{\mathsf{pop}*}$ (or respectively $>_{\mathsf{pop}}$) is applicable. Thus $\pi(t)$ is a subterm of $\pi(s)$ modulo the equivalence $\approx$. We conclude $\pi(t\sigma) \in \mathsf{Val}_\pi$, and hence $\mathsf{S}_\pi(s\sigma) = [\,] = \mathsf{S}_\pi(t\sigma)$. Finally, note that $\|\pi(s\sigma)\| > \|\pi(t\sigma)\|$ as $\pi(t\sigma)$ is a subterm of $\pi(s\sigma)$. Thus $\mathsf{N}_\pi(s\sigma) \blacktriangleright_1 \mathsf{N}_\pi(t\sigma)$ follows as well. □

To improve the clarity of the exposition, we concentrate on the crucial cases in the proofs of the following lemmas. The interested reader is kindly referred to [17] for the full proof.

Lemma 18. *Suppose* $s >^{\pi}_{\mathsf{pop}} t$ *such that* $\pi(s\sigma) = f(\pi(s_1\sigma), \dots, \pi(s_n\sigma))$ *with* $\pi(s_i\sigma) \in \mathsf{Val}_\pi$ *for* $i \in \{1, \dots, n\}$. *Moreover suppose* $\mathsf{nrm}(f) = \{j_1, \dots, j_q\}$. *Then* $f^{\mathsf{n}}(\mathsf{N}_\pi(s_{j_1}\sigma), \dots, \mathsf{N}_\pi(s_{j_q}\sigma)) \succcurlyeq_{3\cdot\|\pi(t)\|} \mathsf{N}_\pi(t\sigma)$ *holds.*

Proof. Note that the assumption implies that the argument filtering π does not collapse f. We show the lemma by induction on $>^{\pi}_{\mathsf{pop}}$. We consider the subcase that $s >^{\pi}_{\mathsf{pop}} t$ follows as $t = g(t_1, \dots, t_m)$, π does not collapse on g, $f \succ g$, and $s >^{\pi}_{\mathsf{pop}} t_j$ for all $j \in \pi(g)$, cf. Definition 10(2). We set $u := f^{\mathsf{n}}(\mathsf{N}_\pi(s_{j_1}\sigma), \dots, \mathsf{N}_\pi(s_{j_q}\sigma))$ and $k := 3 \cdot \|\pi(t)\|$ and first prove $u \succcurlyeq_{k-1} \mathsf{S}_\pi(t\sigma)$.

If $\pi(t\sigma) \in \mathsf{Val}_\pi$, then $\mathsf{S}_\pi(t\sigma) = [\,]$ is minimal with respect to $\succcurlyeq_{k-1}$. Thus we are done. Hence suppose $\mathsf{nrm}(g) = \{j'_1, \dots, j'_q\}$, $\mathsf{safe}(g) = \{i'_1, \dots, i'_p\}$ and let

$$\mathsf{S}_\pi(t\sigma) = [g^{\mathsf{n}}(\mathsf{N}_\pi(t_{j'_1}\sigma), \dots, \mathsf{N}_\pi(t_{j'_q}\sigma))\ \mathsf{S}_\pi(t_{i'_1}\sigma) \cdots \mathsf{S}_\pi(t_{i'_p}\sigma)]\ .$$

We set $v := g^{\mathsf{n}}(\mathsf{N}_\pi(t_{j'_1}\sigma), \dots, \mathsf{N}_\pi(t_{j'_q}\sigma))$. It suffices to show $u \succcurlyeq_{k-2} v$ and $u \succcurlyeq_{k-2} \mathsf{S}_\pi(t_j\sigma)$ for $j \in \mathsf{safe}(g)$. Both assertions follow from the induction hypothesis.

Now consider $\mathsf{N}_\pi(t\sigma) = [\mathsf{S}_\pi(t\sigma)\ \mathsf{s} \cdots \mathsf{s}]$ with $\|\pi(t\sigma)\|$ occurrences of the constant s. Recall that $\mathsf{width}(\mathsf{N}_\pi(t\sigma)) = \|\pi(t\sigma)\| + 1$. Observe that $f^{\mathsf{n}} \succ \mathsf{s}$. Hence to prove $u \succcurlyeq_k \mathsf{S}_\pi(t\sigma)$ it suffices to observe that $\mathsf{width}(u) + k > \|\pi(t\sigma)\| + 1$ holds. For that note that $\|\pi(t\sigma)\|$ is either $\|\pi(t_j\sigma)\| + 1$ for some $j \in \pi(g)$ or less than k. In the latter case, we are done. Otherwise $\|\pi(t\sigma)\| = \|\pi(t_j\sigma)\| + 1$. Then from the definition of $\succcurlyeq_k$ and the induction hypothesis $u \succcurlyeq_{3\cdot\|\pi(t_j)\|} \mathsf{N}_\pi(t_j\sigma)$ we can conclude $\mathsf{width}(u) + 3 \cdot \|\pi(t_j)\| > \mathsf{width}(\mathsf{N}_\pi(t_j\sigma)) = \|\pi(t_j\sigma)\| + 1$. Since $k \geqslant 3 \cdot (\|\pi(t_j)\| + 1)$, $\mathsf{width}(u) + k > \|\pi(t\sigma)\| + 1$ follows. □

Lemma 19. *Suppose* $s >^{\pi}_{\mathsf{pop}*} t$ *such that* $\pi(s\sigma) = f(\pi(s_1\sigma), \ldots, \pi(s_n\sigma))$ *with* $\pi(s_i\sigma) \in \mathsf{Val}_\pi$ *for* $i \in \{1, \ldots, n\}$. *Then for* $\mathsf{nrm}(f) = \{j_1, \ldots, j_q\}$,

(1) $f^{\mathsf{n}}(\mathsf{N}_\pi(s_{j_1}\sigma), \ldots, \mathsf{N}_\pi(s_{j_q}\sigma)) \blacktriangleright_{3\cdot\|\pi(t)\|} \mathsf{S}_\pi(t\sigma)$, *and*
(2) $f^{\mathsf{n}}(\mathsf{N}_\pi(s_{j_1}\sigma), \ldots, \mathsf{N}_\pi(s_{j_q}\sigma)) :: \mathsf{BN}_\pi(s\sigma) \blacktriangleright_{3\cdot\|\pi(t)\|} \mathsf{N}_\pi(t\sigma)$.

Proof. The lemma is shown by induction on the definition of $>^{\pi}_{\mathsf{pop}*}$. Set $u = f^{\mathsf{n}}(\mathsf{N}_\pi(s_{j_1}\sigma), \ldots, \mathsf{N}_\pi(s_{j_q}\sigma))$. Suppose $s >^{\pi}_{\mathsf{pop}*} t$ follows due to Definition 11(2). We set $k := 3 \cdot \|\pi(t)\|$. Let $\mathsf{nrm}(g) = \{j'_1, \ldots, j'_q\}$ and let $\mathsf{safe}(g) = \{i'_1, \ldots, i'_p\}$.

Property (1) is immediate for $\pi(t\sigma) \in \mathsf{Val}_\pi$, so assume otherwise. We see that $s >^{\pi}_{\mathsf{pop}} t_j$ for all $j \in \mathsf{nrm}(g)$ and obtain $u \succcurlyeq_{k-1} g^{\mathsf{n}}(\mathsf{N}_\pi(t_{j'_1}\sigma), \ldots, \mathsf{N}_\pi(t_{j'_q}\sigma))$ as in Lemma 18. Furthermore, $s >^{\pi}_{\mathsf{pop}*} t_{j_0}$ for some $j_0 \in \mathsf{safe}(g)$ and by induction hypothesis: $u \blacktriangleright_{k-1} \mathsf{S}_\pi(t_{j_0}\sigma)$. To conclude property (1), it remains to verify $u \succcurlyeq_{k-1} \mathsf{S}_\pi(t_j\sigma)$ for the remaining $j \in \mathsf{safe}(g)$. We either have $s >^{\pi}_{\mathsf{pop}} t_j$ or $\pi(s_i) \trianglerighteq \pi(t_j)$ (for some i). In the former subcase we proceed as in the claim, and for the latter we observe $\pi(t_j\sigma) \in \mathsf{Val}_\pi$, and thus $\mathsf{S}_\pi(t_j\sigma) = [\,]$ follows. This establishes property (1).

To conclude property (2), it suffices to show $\mathsf{width}(u :: \mathsf{BN}_\pi(s\sigma)) + k > \mathsf{width}(\mathsf{N}_\pi(t\sigma))$, or equivalently $\|\pi(s\sigma)\| + 1 + k > \|\pi(t\sigma)\|$. The latter can be shown, if we proceed similar as in the claim. □

Recall the definition of $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ from Section 2 and define $\mathcal{Q} := \{f(x_1, \ldots, x_n) \to \bot \mid f \in \mathcal{D}\}$, and set $\xrightarrow{\mathsf{v}}_{\mathcal{R}} := \xrightarrow{\mathcal{Q}}_{\mathcal{R}}$. We suppose $\bot \in \mathcal{F}$ is a constructor symbol not occurring in $\mathcal{R}$. As the normal forms of $\mathcal{Q}$ coincide with Val, $\xrightarrow{\mathsf{v}}_{\mathcal{R}}$ is the restriction of $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$, where arguments need to be values instead of normal forms of $\mathcal{R}$. From Lemma 17 and 19 we derive an embedding of root steps $\xrightarrow{\mathsf{v}}^{\varepsilon}_{\mathcal{R}}$.

Now, suppose the step $s \xrightarrow{\mathsf{v}}_{\mathcal{R}} t$ takes place below the root. Observe that $\pi(s) \neq \pi(t)$ need not hold in general. Thus we cannot hope to prove $\mathsf{N}_\pi(s) \blacktriangleright_k \mathsf{N}_\pi(t)$. However, we have the following stronger result.

Lemma 20. *There exists a uniform* $k \in \mathbb{N}$ *(depending only on* $\mathcal{R}$*) such that if* $\mathcal{R} \subseteq >^{\pi}_{\mathsf{pop}*}$ *holds then* $s \xrightarrow{\mathsf{v}}^{\varepsilon}_{\mathcal{R}} t$ *implies* $\mathsf{N}_\pi(s) \blacktriangleright_k \mathsf{N}_\pi(t)$. *Moreover, if* $\mathcal{R} \subseteq \gtrsim^{\pi}_{\mathsf{pop}*}$ *holds then* $s \xrightarrow{\mathsf{v}}_{\mathcal{R}} t$ *implies* $\mathsf{N}_\pi(s) \blacktriangleright\!\!\sim_k \mathsf{N}_\pi(t)$.

Proof. We consider the first half of the assertion. Suppose $\mathcal{R} \subseteq >^{\pi}_{\mathsf{pop}*}$ and $s \xrightarrow{\mathsf{v}}^{\varepsilon}_{\mathcal{R}} t$, that is for some rule $f(l_1, \ldots, l_n) \to r \in \mathcal{R}$ and substitution $\sigma : \mathcal{V} \to \mathsf{Val}$ we have $s = f(l_1\sigma, \ldots, l_n\sigma)$ and $t = r\sigma$. Depending on whether π collapses f, the property either directly follows from Lemma 17 or is a consequence of Lemma 19(2).

In order to conclude the second half of the assertion, one performs induction on the rewrite context. In addition, one shows that for the special case $\mathsf{S}_\pi(s) \approx \mathsf{S}_\pi(t)$, still $\|\pi(s)\| \geqslant \|\pi(t)\|$ holds. From this the lemma follows. □

For constructor TRSs, we can simulate $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ using $\xrightarrow{\mathsf{v}}_{\mathcal{R}}$. We extend $\mathcal{R}$ with suitable rules $\Phi(\mathcal{R})$, which replace normal forms that are not values by some constructor symbol. To simplify the argument we re-use the symbol $\bot$ from above. We define the TRS $\Phi(\mathcal{R})$ as

$$\Phi(\mathcal{R}) := \{f(t_1, \ldots, t_n) \to \bot \mid f(t_1, \ldots, t_n) \in \mathsf{NF}(\mathcal{R}) \cap \mathcal{T}(\mathcal{F}) \text{ and } f \in \mathcal{D}\}\ .$$

Moreover, we define $\phi_{\mathcal{R}}(t) := t{\downarrow_{\Phi(\mathcal{R})}}$. Observe that $\phi_{\mathcal{R}}(\cdot)$ is well-defined since $\Phi(\mathcal{R})$ is confluent and terminating.

Lemma 21. *Let $\mathcal{R} \cup \mathcal{S}$ be a constructor TRS. Define $\mathcal{S}' := \mathcal{S} \cup \Phi(\mathcal{R} \cup \mathcal{S})$. For $s \in \mathcal{T}(\mathcal{F})$,*

$$s \xrightarrow{\mathsf{i}}{}^{\varepsilon}_{\mathcal{R}/\mathcal{S}} t \quad \textit{implies} \quad \phi_{\mathcal{R}\cup\mathcal{S}}(s) \xrightarrow{\mathsf{v}}{}^{\varepsilon}_{\mathcal{R}/\mathcal{S}'} \phi_{\mathcal{R}\cup\mathcal{S}}(t) \;,$$

where $\xrightarrow{\mathsf{v}}_{\mathcal{R}/\mathcal{S}'}$ abbreviates $\xrightarrow{\mathsf{v}}{}^{*}_{\mathcal{S}'} \cdot \xrightarrow{\mathsf{v}}_{\mathcal{R}} \cdot \xrightarrow{\mathsf{v}}{}^{*}_{\mathcal{S}'}$.

Proof. It is easy to see that $s \xrightarrow{\mathsf{i}}_{\mathcal{R}} t$ implies $\phi_{\mathcal{R}}(s) \xrightarrow{\mathsf{v}}_{\mathcal{R}} \cdot \xrightarrow{\mathsf{v}}{}^{!}_{\Phi(\mathcal{R})} \phi_{\mathcal{R}}(t)$. Suppose $s \xrightarrow{\mathsf{i}}{}^{\varepsilon}_{\mathcal{R}/\mathcal{S}} t$, then there exist ground terms u and v such that $s \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{S}} u \xrightarrow{\mathsf{i}}{}^{\varepsilon}_{\mathcal{R}} v \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{S}} t$. Let $\phi(t) := \phi_{\mathcal{R}\cup\mathcal{S}}(t)$. From the above, $\phi(s) \xrightarrow{\mathsf{v}}{}^{*}_{\mathcal{S}'} \phi(u) \xrightarrow{\mathsf{v}}{}^{\varepsilon}_{\mathcal{R}} \cdot \xrightarrow{\mathsf{v}}{}^{*}_{\mathcal{S}'} \phi(v) \xrightarrow{\mathsf{v}}{}^{*}_{\mathcal{S}'} \phi(t)$ follows as desired. □

Suppose $\mathcal{R} \subseteq >^{\pi}_{\mathsf{pop}*}$ and $\mathcal{S} \subseteq \gtrsim^{\pi}_{\mathsf{pop}*}$ holds. Together with Lemma 20, the above simulation establishes the promised embedding of $\xrightarrow{\mathsf{i}}{}^{\varepsilon}_{\mathcal{R}/\mathcal{S}}$ into $\blacktriangleright_k$.

Lemma 22. *Let $\mathcal{R} \cup \mathcal{S}$ be a constructor TRS, and suppose $\mathcal{R} \subseteq >^{\pi}_{\mathsf{pop}*}$ and $\mathcal{S} \subseteq \gtrsim^{\pi}_{\mathsf{pop}*}$ hold. Then for k depending only on $\mathcal{R}$ and $\mathcal{S}$, we have for $s \in \mathcal{T}(\mathcal{F})$,*

$$s \xrightarrow{\mathsf{i}}{}^{\varepsilon}_{\mathcal{R}/\mathcal{S}} t \quad \textit{implies} \quad \mathsf{N}_{\pi}(\phi(s)) \blacktriangleright^{+}_{k} \mathsf{N}_{\pi}(\phi(t)) \;.$$

Proof. Consider a step $s \xrightarrow{\mathsf{i}}{}^{\varepsilon}_{\mathcal{R}/\mathcal{S}} t$ and set $\phi(t) := \phi_{\mathcal{R}\cup\mathcal{S}}(t)$. By Lemma 21 there exist terms u and v such that $\phi(s) \xrightarrow{\mathsf{v}}{}^{*}_{\mathcal{S}\cup\Phi(\mathcal{R}\cup\mathcal{S})} u \xrightarrow{\mathsf{v}}{}^{\varepsilon}_{\mathcal{R}} v \xrightarrow{\mathsf{v}}{}^{*}_{\mathcal{S}\cup\Phi(\mathcal{R}\cup\mathcal{S})} \phi(t)$. Since $\mathcal{R} \subseteq >^{\pi}_{\mathsf{pop}*}$ holds, by Lemma 20 $\mathsf{N}_{\pi}(u) \blacktriangleright_{k_1} \mathsf{N}_{\pi}(v)$ follows. Moreover from $\mathcal{S} \subseteq \gtrsim^{\pi}_{\mathsf{pop}*}$ together with Lemma 20 we conclude that $r_1 \xrightarrow{\mathsf{v}}_{\mathcal{S}\cup\Phi(\mathcal{R}\cup\mathcal{S})} r_2$ implies $\mathsf{N}_{\pi}(r_1) \blacktriangleright_{\sim k_2} \mathsf{N}_{\pi}(r_2)$. Here we use the easily verified fact that steps using $\Phi(\mathcal{R} \cup \mathcal{S})$ are embeddable into $\blacktriangleright_{\sim k_2}$. In both cases k_1 and k_2 depend only on $\mathcal{R}$ and $\mathcal{S}$ respectively; set $k := \max\{k_1, k_2\}$. In sum we have $\mathsf{N}_{\pi}(\phi(s)) \blacktriangleright^{*}_{\sim k} \mathsf{N}_{\pi}(u) \blacktriangleright_k \mathsf{N}_{\pi}(v) \blacktriangleright^{*}_{\sim k} \mathsf{N}_{\pi}(\phi(t))$. It is an easy to see that $\blacktriangleright_k \cdot \approx \subseteq \blacktriangleright_k$ and $\approx \cdot \blacktriangleright_k \subseteq \blacktriangleright_k$ holds. Hence the lemma follows. □

Theorem 23. *Let $\mathcal{R} \cup \mathcal{S}$ be a constructor TRS, and suppose $\mathcal{R} \subseteq >^{\pi}_{\mathsf{pop}*}$ and $\mathcal{S} \subseteq \gtrsim^{\pi}_{\mathsf{pop}*}$ holds. Then there exists a polynomial p depending only $\mathcal{R} \cup \mathcal{S}$ such that for any basic term t, $\mathrm{dl}(t, \xrightarrow{\mathsf{i}}{}^{\varepsilon}_{\mathcal{R}/\mathcal{S}}) \leqslant p(|t|)$.*

Proof. Assume $t \notin \mathsf{NF}(\mathcal{R}\cup\mathcal{S})$, otherwise $\mathrm{dl}(t, \xrightarrow{\mathsf{i}}{}^{\varepsilon}_{\mathcal{R}/\mathcal{S}})$ is trivially bounded. Without loss of generality, we assume that t is ground. As t is a basic term: $\phi_{\mathcal{R}\cup\mathcal{S}}(t) = t$. From Lemma 22 we infer (for some k) $\mathrm{dl}(t, \xrightarrow{\mathsf{i}}{}^{\varepsilon}_{\mathcal{R}/\mathcal{S}}) \leqslant \mathsf{G}_k(\mathsf{N}_{\pi}(\phi_{\mathcal{R}\cup\mathcal{S}}(t))) = \mathsf{G}_k(\mathsf{N}_{\pi}(t))$, such that the latter is polynomially bounded in $|t|$ and the polynomial only depends on k, cf. Lemma 16. Note that k depends only on $\mathcal{R} \cup \mathcal{S}$. □

Suppose $\mathcal{R}$ is a constructor TRS, and let $\mathcal{P}$ denote the set of weak innermost dependency pairs. For the moment, suppose that all compound symbols of $\mathcal{P}$ are nullary. Provided that $\mathcal{P}$ is non-duplicating and $\mathcal{U}(\mathcal{P})$ compatible with some SLI, as a consequence of the above theorem paired with Proposition 7, the inclusions $\mathcal{P} \subseteq >^{\pi}_{\mathsf{pop}*}$ and $\mathcal{U}(\mathcal{P}) \subseteq \gtrsim^{\pi}_{\mathsf{pop}*}$ certify that $\mathrm{rc}^{\mathsf{i}}_{\mathcal{R}}$ is polynomially bounded. Observe that for the application of $>^{\pi}_{\mathsf{pop}*}$ and $\gtrsim^{\pi}_{\mathsf{pop}*}$ in the context of $\mathcal{P}$ and $\mathcal{U}(\mathcal{P})$, we alter Definitions 10 and 11 such that $f \in \mathcal{D}^{\sharp}$ is demanded.

Example 24 (Example 14 continued). Reconsider the TRS $\mathcal{R}_{\mathsf{bits}}$, and let $\mathcal{P}$ denote $\mathsf{WIDP}(\mathcal{R}_{\mathsf{bits}})$ as drawn in Example 5. By taking the SLI $\mathcal{A}$ with $0_{\mathcal{A}} = 0$, $\mathsf{s}_{\mathcal{A}}(x) = x{+}1$ and $\mathsf{half}_{\mathcal{A}}(x) = x{+}1$ we obtain $\mathcal{U}(\mathcal{P}) \subseteq >_{\mathcal{A}}$ and moreover, observe that $\mathcal{P}$ is both non-duplicating and contains only nullary compound symbols. In Example 14 we have seen that $\mathcal{P} \subseteq >^{\pi}_{\mathsf{pop}*}$ holds. Similarly, $\mathcal{U}(\mathsf{WIDP}(\mathcal{R}_{\mathsf{bits}})) \subseteq \gtrsim^{\pi}_{\mathsf{pop}*}$ can easily be shown. From the above observation we thus conclude a polynomial runtime complexity of $\mathcal{R}_{\mathsf{bits}}$.

The assumption that all compound symbols from $\mathcal{P}$ need to be nullary is straightforward to lift, but technical. Hence, we do not provide a complete proof here, but only indicate the necessary changes, see [18] for the formal construction.

Note that in the general case, it does not suffice to embed root steps of $\mathcal{P}$ into $\blacktriangleright_k$, rather we have to embed steps of form $C[s^\sharp_1, \ldots, s^\sharp_i, \ldots, s^\sharp_n] \xrightarrow{\mathsf{v}}_{\mathcal{P}} C[s^\sharp_1, \ldots, t^\sharp_i, \ldots, s^\sharp_n]$ with C being a context built from compound symbols. As first measure we require that the argument filtering π is *safe* [5], that is $\pi(c) = [1, \ldots, n]$ for each compound symbol c of arity n. Secondly, we adapt the predicative interpretation N_π in such a way that compound symbols are interpreted as sequences, and their arguments by the interpretation N_π. This way, the renewed embedding requires $\mathsf{N}_\pi(s^\sharp_i) \blacktriangleright_k \mathsf{N}_\pi(t^\sharp_i)$ instead of $\mathsf{S}_\pi(s^\sharp_i) \blacktriangleright_k \mathsf{S}_\pi(t^\sharp_i)$.

Theorem 25. *Let $\mathcal{R}$ be a constructor TRS, and let $\mathcal{P}$ denote the set of weak innermost dependency pairs. Assume $\mathcal{P}$ is non-duplicating, and suppose $\mathcal{U}(\mathcal{P}) \subseteq >_{\mathcal{A}}$ for some SLI $\mathcal{A}$. Let π be a safe argument filtering. If $\mathcal{P} \subseteq >^{\pi}_{\mathsf{pop}*}$ and $\mathcal{U}(\mathcal{P}) \subseteq \gtrsim^{\pi}_{\mathsf{pop}*}$ then the innermost runtime complexity $\mathrm{rc}^{\mathsf{i}}_{\mathcal{R}}$ induced is polynomially bounded.*

Above it is essential that $\mathcal{R}$ is a constructor TRS. This even holds if POP* is applied directly.

Example 26. Consider the TRS $\mathcal{R}_{\mathsf{exp}}$ below:

$$\mathsf{exp}(x) \to \mathsf{e}(\mathsf{g}(x)) \qquad \mathsf{e}(\mathsf{g}(\mathsf{s}(x))) \to \mathsf{dp}_1(\mathsf{g}(x)) \qquad \mathsf{g}(\mathsf{0}) \to \mathsf{0}$$

$$\mathsf{dp}_1(x) \to \mathsf{dp}_2(\mathsf{e}(x), x) \qquad \mathsf{dp}_2(x, y) \to \mathsf{pr}(x, \mathsf{e}(y))$$

The above rules are oriented directly by $>_{\mathsf{pop}*}$ induced by safe and $\succsim$ such that: (i) the argument position of g and exp are normal, the remaining argument positions are safe, and (ii) $\mathsf{exp} \succ \mathsf{g} \succ \mathsf{dp}_1 \succ \mathsf{dp}_2 \succ \mathsf{e} \succ \mathsf{pr} \succ \mathsf{0}$. On the other hand, $\mathcal{R}_{\mathsf{exp}}$ admits at least exponential innermost runtime-complexity, as for instance $\mathsf{exp}(s^n(\mathsf{0}))$ normalizes in exponentially (in n) many innermost rewrite steps.

We adapt the definition of $>_{\mathsf{pop}*}$ in the sense that we refine the notion of defined function symbols as follows. Let $\mathcal{G}_{\mathcal{C}}$ denote the least set containing $\mathcal{C}$ and all symbols appearing in arguments to left-hand sides in $\mathcal{R}$. Moreover, set $\mathcal{G}_{\mathcal{D}} := \mathcal{F} \setminus \mathcal{G}_{\mathcal{C}}$ and set $\mathsf{Val} := \mathcal{T}(\mathcal{G}_{\mathcal{C}}, \mathcal{V})$. Then in order to extend Theorem 25 to non-constructor TRS it suffices to replace $\mathcal{D}$ by $\mathcal{G}_{\mathcal{D}}$ and $\mathcal{C}$ by $\mathcal{G}_{\mathcal{C}}$ in all above given definitions and arguments (see [17] for the formal construction). Thus the next theorem follows easily from combining Proposition 9 and Theorem 25. This theorem can be extended so that in each path different termination techniques (inducing polynomial runtime complexity) are employed, see [6] and Section 6.

Theorem 27. *Let $\mathcal{R}$ be a TRS. Let $\mathcal{G}$ denote the weak innermost dependency graph, and let $\mathcal{F} = \mathcal{G}_\mathcal{D} \uplus \mathcal{G}_\mathcal{C}$ be separated as above. Suppose for every path $(\mathcal{P}_1, \dots, \mathcal{P}_n)$ in $\mathcal{G}/_\equiv$ there exists an SLI $\mathcal{A}$ and a pair $(\gtrsim^\pi_{\mathsf{pop}*}, >^\pi_{\mathsf{pop}*})$ based on a safe argument filtering π such that (i) $\mathcal{U}(\mathcal{P}_1 \cup \dots \cup \mathcal{P}_n) \subseteq >_\mathcal{A}$ (ii) $\mathcal{P}_1 \cup \dots \cup \mathcal{P}_{n-1} \cup \mathcal{U}(\mathcal{P}_1 \cup \dots \cup \mathcal{P}_n) \subseteq \gtrsim^\pi_{\mathsf{pop}*}$, and (iii) $\mathcal{P}_n \subseteq >^\pi_{\mathsf{pop}*}$ holds. Then the innermost runtime complexity $\mathrm{rc}^{\mathsf{i}}_\mathcal{R}$ induced is polynomially bounded.*

The next theorem establishes that POP* in conjunction with (weak innermost) dependency pairs induces polytime computability of the function described through the analysed TRS. We kindly refer the reader to [18] for the proof.

Theorem 28. *Let $\mathcal{R}$ be an orthogonal, S-sorted and completely defined constructor TRS such that the underlying signature is simple. Let $\mathcal{P}$ denote the set of weak innermost dependency pairs. Assume $\mathcal{P}$ is non-duplicating, and suppose $\mathcal{U}(\mathcal{P}) \subseteq >_\mathcal{A}$ for some SLI $\mathcal{A}$. If $\mathcal{P} \subseteq >^\pi_{\mathsf{pop}*}$ and $\mathcal{U}(\mathcal{P}) \subseteq \gtrsim^\pi_{\mathsf{pop}*}$ then the functions computed by $\mathcal{R}$ are computable in polynomial time.*

Here *simple* signature [19] essentially means that the size of any constructor term depends polynomially on its depth. Such a restriction is always necessary in this context, compare [18] and e.g. [19]. This restriction is also responsible for the introduction of sorts.

6 Experimental Results

All described techniques have been incorporated into the *Tyrolean Complexity Tool* TCT, an open source complexity analyser[1]. We performed tests on two testbeds: **T** constitutes of the 1394 examples from the Termination Problem Database Version 5.0.2 used in the runtime complexity category of the termination competition 2008[2]. Moreover, testbed **C** is the restriction of testbed **T** to constructor TRSs (638 in total). All experiments were conducted on a machine that is identical to the official competition server (8 AMD Opteron® 885 dual-core processors with 2.8GHz, 8x8 GB memory). As timeout we use 5 seconds. We orient TRSs using $>^\pi_{\mathsf{pop}*}$ by encoding the constraints on precedence and so forth in *propositional logic* (cf. [17] for details), employing MiniSat [20] for finding satisfying assignments. In a similar spirit, we check compatibility with SLIs via translations to SAT. In order to derive an estimated dependency graph, we use the function ICAP (cf. [21]).

Experimental findings are summarised in Table 1.[3] In each column, we highlight the total on yes-, maybe- and timeout-instances. Furthermore, we annotate average times in seconds. In the first three columns we contrast POP* as direct technique to POP* as base to (weak innermost) dependency pairs. I.e., the columns WIDP and WIDG show results concerning Proposition 7 together with Theorem 25 or Theorem 27 respectively.

[1] Available at `http://cl-informatik.uibk.ac.at/software/tct`

[2] See `http://termcomp.uibk.ac.at`

[3] See `http://cl-informatik.uibk.ac.at/~zini/rta09` for extended results.

Table 1. Experimental Results

		polynomial path orders			dependency graphs mixed			
		DIRECT	WIDP	WIDG	P	PP	M	MP
T	Yes	46/0.03	69/0.09	80/0.07	198/0.54	198/0.51	200/0.63	207/0.48
	Maybe	1348/0.04	1322/0.10	1302/0.14	167/0.77	170/0.82	142/0.61	142/0.63
	Timeout	0	3	12	1029	1026	1052	1045
C	Yes	40/0.03	48/0.08	55/0.05	99/0.40	100/0.38	98/0.26	105/0.23
	Maybe	598/0.05	587/0.10	576/0.13	143/0.72	146/0.77	119/0.51	119/0.54
	Timeout	0	3	7	396	392	421	414

In the remaining four columns we assess the power of Proposition 7 and 9 in conjunction with different base orders, thus verifying that the use of POP* in this context is independent to existing techniques. Column P asserts that the different paths are handled by *linear and quadratic restricted interpretations* [5]. In column PP, in addition POP* is employed. Similar, in column M *restricted matrix interpretations* (that is matrix interpretations [22], where constructors are interpreted by triangular matrices) are used to handle different paths. Again column MP extends column M with POP*. Note that all methods induce polynomial innermost runtime complexity.

Table 1 reflects that the integration of POP* in the context of (weak) dependency pairs, significantly extends the direct approach. Worthy of note, the extension of [2] with quasi-precedences alone gives 5 additional examples. As advertised, POP* is incredibly fast in all settings. Consequently, as evident from the table, polynomial path orders team well with existing techniques, without affecting overall performance: note that due to the addition of POP* the number of timeouts is reduced.

7 Conclusion

In this paper we study the runtime complexity of rewrite systems. We combine two recently developed techniques in the context of complexity analysis: weak innermost dependency pairs and polynomial path orders. If the conditions of our main result are met, we can conclude the innermost polynomial runtime complexity of the studied term rewrite system. And we obtain that the function defined are *polytime computable*. We have implemented the technique and experimental evidence clearly indicates the power and in particular the efficiency of the new method.

References

1. Hofbauer, D., Lautemann, C.: Termination proofs and the length of derivations. In: Dershowitz, N. (ed.) RTA 1989. LNCS, vol. 355, pp. 167–177. Springer, Heidelberg (1989)

2. Avanzini, M., Moser, G.: Complexity analysis by rewriting. In: Garrigue, J., Hermenegildo, M.V. (eds.) FLOPS 2008. LNCS, vol. 4989, pp. 130–146. Springer, Heidelberg (2008)
3. Simmons, H.: The realm of primitive recursion. ARCH 27, 177–188 (1988)
4. Bellantoni, S., Cook, S.: A new recursion-theoretic characterization of the polytime functions. CC 2, 97–110 (1992)
5. Hirokawa, N., Moser, G.: Automated complexity analysis based on the dependency pair method. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 364–379. Springer, Heidelberg (2008)
6. Hirokawa, N., Moser, G.: Complexity, graphs, and the dependency pair method. In: Cervesato, H. (ed.) LPAR 2008. LNCS, vol. 5330, pp. 667–681. Springer, Heidelberg (2008)
7. Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: SAT solving for termination analysis with polynomial interpretations. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 340–354. Springer, Heidelberg (2007)
8. Geser, A.: Relative Termination. PhD thesis, University of Passau, Faculty for Mathematics and Computer Science (1990)
9. Bonfante, G., Marion, J.Y., Moyen, J.Y.: Quasi-interpretations: A way to control resources. In: TCS (to appear, 2009)
10. Marion, J.Y., Péchoux, R.: Characterizations of polynomial complexity classes with a better intensionality. In: PPDP 2008, pp. 79–88. ACM, New York (2008)
11. Arai, T., Moser, G.: Proofs of termination of rewrite systems for polytime functions. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 529–540. Springer, Heidelberg (2005)
12. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
13. Terese: Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press, Cambridge (2003)
14. Thiemann, R.: The DP Framework for Proving Termination of Term Rewriting. PhD thesis, University of Aachen, Department of Computer Science (2007)
15. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. TCS 236, 133–178 (2000)
16. Avanzini, M., Moser, G., Schnabl, A.: Automated implicit computational complexity analysis (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 132–138. Springer, Heidelberg (2008)
17. Avanzini, M.: Automation of polynomial path orders. Master's thesis, University of Innsbruck, Faculty for Computer Science (2009), `http://cl-informatik.uibk.ac.at/~zini/MT.pdf`
18. Avanzini, M., Moser, G.: Dependency pairs and polynomial path orders. CoRR ab/cs/0904.0981 (2009), `http://www.arxiv.org/`
19. Marion, J.Y.: Analysing the implicit complexity of programs. IC 183, 2–18 (2003)
20. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
21. Giesl, J., Thiemann, R., Schneider-Kamp, P.: Proving and disproving termination of higher-order functions. In: Gramlich, B. (ed.) FroCos 2005. LNCS, vol. 3717, pp. 216–231. Springer, Heidelberg (2005)
22. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of term rewriting. JAR 40, 195–220 (2008)

Unique Normalization for Shallow TRS

Guillem Godoy[1,⋆] and Florent Jacquemard[2]

[1] Technical University of Catalonia, Jordi Girona 1, Barcelona, Spain
ggodoy@lsi.upc.edu
[2] INRIA Saclay & LSV (UMR CNRS/ENS Cachan), France
florent.jacquemard@inria.fr

Abstract. Computation with a term rewrite system (TRS) consists in the application of its rules from a given starting term until a normal form is reached, which is considered the result of the computation. The unique normalization (UN) property for a TRS $\mathcal{R}$ states that any starting term can reach at most one normal form when $\mathcal{R}$ is used, i.e. that the computation with $\mathcal{R}$ is unique.

We study the decidability of this property for classes of TRS defined by syntactic restrictions such as linearity (variables can occur only once in each side of the rules), flatness (sides of the rules have height at most one) and shallowness (variables occur at depth at most one in the rules).

We prove that UN is decidable in polynomial time for shallow and linear TRS, using tree automata techniques. This result is very near to the limits of decidability, since this property is known undecidable even for very restricted classes like right-ground TRS, flat TRS and also right-flat and linear TRS. We also show that UN is even undecidable for flat and right-linear TRS. The latter result is in contrast with the fact that many other natural properties like reachability, termination, confluence, weak normalization, etc. are decidable for this class of TRS.

Introduction

Term rewriting is a Turing-complete model of computation. Computation with a TRS consists in the application of its rules from a given starting term until a normal form is reached, i.e. a term that cannot be rewritten any more, which is usually considered as the result of the computation.

The unique normalization (UN) property for a TRS $\mathcal{R}$ states that any starting term can reach at most one normal form when $\mathcal{R}$ is used, i.e. that the computation with $\mathcal{R}$ is unique. This property is hence very desirable when dealing with TRS as computation models, and therefore it is important to establish the borders of its decidability.

Other interesting and much studied properties of TRS are reachability (whether a given term can be derived with a given TRS from another given term),

⋆ The first author was supported by Spanish Min. of Educ. and Science by the FORMALISM project (TIN2007-66523) and by the LOGICTOOLS-2 project (TIN2007-68093-C02-01).

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 63–77, 2009.
© Springer-Verlag Berlin Heidelberg 2009

joinability (whether two given terms can be rewritten into one common term), termination (whether there are no infinite derivations from any starting term), confluence (whether every two terms derived from a common one can also be rewritten into another common term), weak normalization (whether every term can be rewritten into a normal form), uniqueness of normal forms (whether every term has at most one normal form equivalent to it modulo the equational theory induced by the TRS), etc. Note that a TRS may satisfy the unique normalization property and simultaneously unsatisfy the uniqueness of normal forms property. This is the case of $\mathcal{R} = \{a \to b,\ a \to c,\ c \to c,\ d \to c,\ d \to e\}$, where all terms can reach at most one normal form, but it holds that the normal forms b and c are equivalent.

In the recent years there has been much progress on determining decidability of these fundamental properties for several classes of TRS, which are defined by imposing certain syntactic restrictions on the rules. Some of the restrictions usually taken into consideration are groundness (no variable appears in the rules), linearity (variables can occur only once in each side of the rules), flatness (sides of the rules have height at most one) and shallowness (variables occur at depth at most one in the rules). When these restrictions refer only to one side of the rules, then we talk about left-linearity, right-linearity, left-flatness, etc.

Some of the strongest known decidability results are the following. Reachability and joinability are decidable for right-shallow right-linear TRS [11], and even for weaker restrictions [15]. Termination is decidable for right-shallow right-linear TRS [5] and other variants of syntactic restrictions based on the form of the dependency pairs obtained from a TRS [20]. Confluence is decidable for shallow right-linear TRS [8], and for right-(ground or variable) TRS [7]. The weak normalization problem is decidable for left-shallow left-linear TRS [11], right-shallow linear TRS and shallow right-linear TRS [6]. Uniqueness of normal forms is decidable in polynomial time for linear shallow TRS [19].

On the negative side, all of these properties have been proved undecidable for flat TRS [9,10,5,4].

The case of the UN property seems to be more difficult. In [18], a polynomial time algorithm is given for UN and TRS with ground rules. On the negative side, UN is undecidable for right-flat and linear TRS [6], for right-ground TRS [16] and for flat TRS [4]. UN has also been shown undecidable for TRS whose rules have at most height two and, moreover, they are left-flat, right-linear, noncollapsing, or linear and noncollapsing in [17]. In [6] the decidability of this problem is left open for flat right-linear TRS.

In this paper, we provide a polynomial time algorithm for deciding UN for shallow and linear TRS (Section 2). We also prove (Section 3) undecidability of UN for flat and right-linear TRS. Our approach for decidability in polynomial time consists in giving a certain characterization of UN. We essentially show that UN is equivalent to the fact that certain regular sets of terms can reach at most one normal form. This characterization can be checked using tree automata techniques. The proof of undecidability is an adequate adaptation of the reductions appearing in [5,4].

1 Preliminaries

Terms Algebra. We use standard notation from the term rewriting literature [1]. A *signature* is a finite set $\Sigma = \bigcup_{i=0}^{max_\Sigma} \Sigma_i$ of function symbols, with i being the arity of symbols in Σ_i. Function symbols of arity 0 are called *constants*. Sometimes we denote a signature as $\{f_1 : a_1, \ldots, f_n : a_n\}$ where each f_i is a function symbol, and each a_i is its corresponding arity. Let $\mathcal{V}$ be a set disjoint from Σ whose elements are called *variables*. The set $\mathcal{T}(\Sigma, \mathcal{V})$ of terms over Σ and $\mathcal{V}$ is defined to be the smallest set containing $\mathcal{V}$ and such that $f(t_1, \ldots, t_m) \in \mathcal{T}(\Sigma, \mathcal{V})$ whenever $f \in \Sigma_m$ and $t_1, \ldots, t_m \in \mathcal{T}(\Sigma, \mathcal{V})$. $\mathcal{V}ar(t)$ denotes the set of variables occurring in the term t.

The *size* $\|t\|$ of a term t is the number of occurrences of variables and function symbols in t. The *height* of a term t, denoted as $height(t)$, is 0 if t is a constant or a variable, and $1 + max\{height(t_1), \ldots, height(t_m)\}$ if $t = f(t_1, \ldots, t_m)$. The positions of a term t, denoted p, q, are sequences of natural numbers that are used to identify the location of subterms of t. The set $\mathcal{P}os(t)$ of *positions* of t is defined by $\mathcal{P}os(t) = \{\epsilon\}$ if t is a constant or a variable, and $\mathcal{P}os(t) = \{\epsilon\} \cup \{1.p \mid p \in \mathcal{P}os(t_1)\} \cup \ldots \cup \{m.p \mid p \in \mathcal{P}os(t_m)\}$ if $t = f(t_1, \ldots, t_m)$, where ϵ denotes the empty sequence and $p.q$ denotes the concatenation of p and q. If t is a term and $p \in \mathcal{P}os(t)$ a position, then $t|_p$ is the subterm of t at position p. More formally, $t|_\epsilon = t$ and $f(t_1, \ldots, t_m)|_{i.p} = t_i|_p$. We denote by $t[s]_p$ ($p \in \mathcal{P}os(t)$) the term that is like t except that the subterm $t|_p$ is replaced by s. More formally, $t[s]_\epsilon = s$ and $f(t_1, \ldots, t_m)[s]_{i.p} = f(t_1, \ldots, t_{i-1}, t_i[s]_p, t_{i+1}, \ldots, t_m)$. We can define a partial order $\leq$ on $\mathcal{P}os(t)$ by $p \leq q$ if and only if p is a prefix of q, i.e there is a sequence p' such that $q = p.p'$. We say that two positions p and q are *parallel*, denoted $p \parallel q$, if they are incomparable with respect to $\leq$. Given a position p in a term s, the *depth* of the occurrence $s|_p$ in s is $|p|$. A *substitution* is a mapping $\mathcal{V} \to \mathcal{T}(\Sigma, \mathcal{V})$. Substitutions can also be applied to arbitrary terms by homomorphically extending its application to variables. The application of a substitution σ to a term t, denoted as $\sigma(t)$, is defined as follows for non-variable terms: $\sigma(f(t_1, \ldots, t_m)) = f(\sigma(t_1), \ldots, \sigma(t_m))$. A *variable renaming* is a substitution from variables to variables.

Term Rewriting. An *extended rewrite rule*, denoted $\ell \to r$, is a pair of terms $\ell \in \mathcal{T}(\Sigma, \mathcal{V})$ (the left-hand side) and $r \in \mathcal{T}(\Sigma, \mathcal{V})$ (the right-hand side). When ℓ is not a variable, and every variable occurring in r occurs also in ℓ, $\ell \to r$ is called a *rewrite rule*. An *extended term rewrite system* (extended TRS) $\mathcal{R}$ is a finite set of extended rewrite rules. A *term rewrite system* (TRS) $\mathcal{R}$ is a finite set of rewrite rules. A term s rewrites to t in one step at position p (by an extended TRS $\mathcal{R}$), denoted by $s \xrightarrow[\mathcal{R},p]{} t$, if $s|_p = \sigma(\ell)$ and $t = s[\sigma(r)]_p$, for some $\ell \to r \in R$ and substitution σ. In this case, s is said to be $\mathcal{R}$-*reducible*. Otherwise s is called an $\mathcal{R}$-*normal form*. The set of $\mathcal{R}$-normal forms is denoted by $\mathsf{NF}_\mathcal{R}$. Sometimes we write $s \xrightarrow[\mathcal{R}]{} t$ when p is not important, or $s \xrightarrow[\mathcal{R},\sigma,p]{} t$, for making the used substitution explicit. The transitive closure, and symmetric and transitive closure of $\xrightarrow[\mathcal{R}]{}$ are denoted as $\xrightarrow[\mathcal{R}]{*}$ and $\xleftrightarrow[\mathcal{R}]{*}$. When $s \xrightarrow[\mathcal{R}]{*} t$ we say that t is *reachable* from s, or that s reaches t. When $s \xleftrightarrow[\mathcal{R}]{*} t$ we say that

s and t are *equivalent*. When there exists a term u reachable from s and t, we say that s and t are *joinable*. When there exists a term u reachable from every term in a set S we say that S is *joinable*. Given $L \subseteq \mathcal{T}(\Sigma)$, we denote $\mathcal{R}^*(L) = \{t \mid \exists s \in L, s \xrightarrow[\mathcal{R}]{*} t\}$. The size of $\mathcal{R}$ is $\|\mathcal{R}\| = \sum_{\ell \to r \in \mathcal{R}}(\|\ell\| + \|r\|)$.

A term is *linear* if no variable occurs more than once in it. A term is *shallow* if all variables occur at depth at most one. A term is *flat* if its height is at most one. A rewrite rule $\ell \to r$ is *flat* (*linear*, *shallow*) if ℓ and r are. A rewrite rule $\ell \to r$ is *right-flat* (*right-linear*, *right-shallow*) if r is. A rewrite rule $\ell \to r$ is *left-flat* (*left-linear*, *left-shallow*) if ℓ is. A TRS is *flat* (*linear*, *shallow*) if all its rules are. A TRS is *right-flat* (*right-linear*, *right-shallow*, *left-flat*, *left-linear*, *left-shallow*) if all its rules are. A TRS is *uniquely normalizing*, or satisfies the *unique normalization* (UN) property, if for each term s and each two normal forms t_1 and t_2 reachable from s, $t_1 = t_2$ holds.

Tree Automata. A *tree automaton* (TA) $\mathcal{A}$ over a signature Σ is a tuple $(Q, Q^{\mathsf{f}}, \Delta)$ where Q is a finite set of nullary state symbols, disjoint from Σ, $Q^{\mathsf{f}} \subseteq Q$ is the subset of final states and Δ is a set of ground rewrite rules of the form: $f(q_1, \ldots, q_m) \to q$, or $q_1 \to q$ (ε-transition) where $f \in \Sigma_m$, and $q_1, \ldots, q_m, q \in Q$ (q is called the *target* state of the rule). The size of $\mathcal{A}$ is $\|\mathcal{A}\| = \sum_{f(q_1,\ldots,q_m)\to q \in \Delta}(m+2) + \sum_{q_1 \to q \in \Delta}(2)$.

The language of ground terms *accepted* by a TA $\mathcal{A}$ on Σ in a state q is the set $L(\mathcal{A}, q) := \{t \in \mathcal{T}(\Sigma) \mid t \xrightarrow[\Delta]{*} q\}$. The language of $\mathcal{A}$ is $L(\mathcal{A}) := \bigcup_{q \in Q^{\mathsf{f}}} L(\mathcal{A}, q)$ and a subset of $\mathcal{T}(\Sigma)$ is called *regular* if it is the language of a TA.

We shall use the following classical properties and problems of TA, see [2] for details.

Proposition 1. *Given two TA $\mathcal{A}_1$ and $\mathcal{A}_2$ over the same signature Σ, one can construct in polynomial time two TA recognizing respectively $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ and $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$, whose sizes are respectively linear and quadratic in $\|\mathcal{A}_1\| + \|\mathcal{A}_2\|$.*

We will consider the two following decision problems for TA:

Problem: Emptiness. **Instance:** a TA $\mathcal{A}$; **Question:** $L(\mathcal{A}) = \emptyset$?
Problem: Singleton. **Instance:** a TA $\mathcal{A}$; **Question:** $|L(\mathcal{A})| = 1$?

Proposition 2. *[2] The emptiness problem is decidable in linear time. The singleton problem is decidable in polynomial time.*

2 Decidability of UN for Shallow and Linear TRS

In this section we prove that UN is decidable in polynomial time for flat and linear TRS (Theorem 1), and this result is immediately extended to shallow and linear TRS (Theorem 2). ¿From now on, we assume a fixed signature Σ and consider the maximal arity max_Σ of its function symbols as a constant. Hence, when analysing complexity of deciding the UN problem, this value is not considered as part of the input. This is a common approach when evaluating complexity for TRS problems (see e.g. the computation of a congruence closure

for shallow equations [12], or the time complexity given for certain problems using tree automata techniques [2]) . This is because, in practice, non-constant function symbols are usually fixed and represent relations with small arity or functions with few parameters, while constants are not fixed and represent the input data of the problem.

After proving some technical lemmas about rewrite reduction with flat and linear TRS in Subsection 2.1, we identify in Subsection 2.2 some necessary and sufficient conditions for UN of such TRS. Finally, in Subsection 2.3 we show how the conditions can be decided by reduction to the above tree automata decision problems.

2.1 Preliminary Results

In this subsection, we first present some technical lemmas concerning the rewriting sequences with flat and linear TRS which will be useful in the proof of Theorem 1. They are based on the notion of the *use* of a position and a derivation which is roughly a form of the descendant of the position.

Definition 1. *Let $\mathcal{R}$ be a flat and linear TRS over Σ. Given a derivation $s \xrightarrow[\mathcal{R}]{*} t$ and a position $p \in \mathcal{P}os(s)$, we define $use(p, s \xrightarrow[\mathcal{R}]{*} t)$ recursively on the length of $s \xrightarrow[\mathcal{R}]{*} t$ as follows:*
– If $s|_p \in \Sigma_0$, then $use(p, s \xrightarrow[\mathcal{R}]{*} t) := s|_p$.
– If $s|_p \notin \Sigma_0$ and $s \xrightarrow[\mathcal{R}]{*} t$ has length 0, then $use(p, s \xrightarrow[\mathcal{R}]{*} t)$ is undefined.
– If $s|_p \notin \Sigma_0$ and $s \xrightarrow[\mathcal{R}]{*} t$ is of the form $s \xrightarrow[p_1,\mathcal{R}]{} s' \xrightarrow[\mathcal{R}]{*} t$ for a position p_1 such that $p_1 \geq p$ or $p_1 \parallel p$ then $use(p, s \xrightarrow[\mathcal{R}]{*} t) := use(p, s' \xrightarrow[\mathcal{R}]{*} t)$.
– If $s|_p \notin \Sigma_0$ and $s \xrightarrow[\mathcal{R}]{*} t$ is of the form $s \xrightarrow[p_1,\ell\to r]{} s' \xrightarrow[\mathcal{R}]{*} t$, where $p = p_1.i.p_2$, and $\ell|_i \in \mathcal{V}$, we consider two cases. If $\ell|_i$ does not occur in r, then $use(p, s \xrightarrow[\mathcal{R}]{*} t)$ is undefined. Otherwise, if $\ell|_i = r|_q$ for some $q \in \mathcal{P}os(r)$, then $use(p, s \xrightarrow[\mathcal{R}]{*} t) := use(p_1.q.p_2, s' \xrightarrow[\mathcal{R}]{*} t)$.

Note that all possible cases are considered since $\mathcal{R}$ is flat. Moreover, the last one is well (uniquely) defined since $\mathcal{R}$ is linear.

Example 1. Let us consider the following flat and linear TRS
$\mathcal{R}_1 = \{x + 0 \to x, s(0) \to c_1, x + c_1 \to s(x), x + y \to y + x, s(c_1) \to 0\}$, and the derivation: $\rho_1 := 0 + s(0) \xrightarrow[\mathcal{R}_1]{} 0 + c_1 \xrightarrow[\mathcal{R}_1]{} s(0) \xrightarrow[\mathcal{R}_1]{} c_1$, and let ρ_1' be its subderivation starting with $0 + c_1$. We have $use(1, \rho_1) = 0$ and $use(2, \rho_1) = use(2, \rho_1') = c_1$. ◇

The following three lemmas can be easily proved by induction on the length of the rewrite sequences.

Lemma 1. *For any flat and linear TRS $\mathcal{R}$, derivation $s \xrightarrow[\mathcal{R}]{*} t$, position $p \in \mathcal{P}os(s)$ and all constant c, if $use(p, s \xrightarrow[\mathcal{R}]{*} t) = c$, then $s[c]_p \xrightarrow[\mathcal{R}]{*} t$ and $s|_p \xrightarrow[\mathcal{R}]{*} c$.*

Lemma 2. *For any flat and linear TRS $\mathcal{R}$, derivation $s \xrightarrow[\mathcal{R}]{*} t$, position $p \in \mathcal{P}os(s)$ such that $use(p, s \xrightarrow[\mathcal{R}]{*} t)$ is undefined, and all variable x, either $s[x]_p \xrightarrow[\mathcal{R}]{*} t$, or there exists a position $q \in \mathcal{P}os(t)$ such that $s[x]_p \xrightarrow[\mathcal{R}]{*} t[x]_q$ and $s|_p \xrightarrow[\mathcal{R}]{*} t|_q$.*

Lemma 3. *For any flat and linear TRS $\mathcal{R}$, derivation $s \xrightarrow[\mathcal{R}]{*} t$, position $p \in \mathcal{P}os(s)$ such that $s|_p$ is a certain variable x, and all (new) variable w not occurring in s, either $s[w]_p \xrightarrow[\mathcal{R}]{*} t$ or there exists a position $q \in \mathcal{P}os(t)$ such that $t|_q = x$ and $s[w]_p \xrightarrow[\mathcal{R}]{*} t[w]_q$.*

2.2 Necessary and Sufficient Conditions for UN

Definition 2. *Let $\mathcal{R}$ be a flat and linear TRS over Σ. A* fork *of $\mathcal{R}$ is a pair of terms $\langle f(s_1, \ldots, s_n), f(t_1, \ldots, t_n)\rangle$, for a function symbol $f \in \Sigma_n$ $(n \geq 0)$, such that for all $i \in \{1, \ldots, n\}$, either s_i and t_i are the same variable of $\mathcal{V}$, or s_i is a constant of Σ_0 and $t_i \xrightarrow[\mathcal{R}]{*} s_i$, or t_i is a constant of Σ_0 and $s_i \xrightarrow[\mathcal{R}]{*} t_i$.*

Example 2. $\langle x + (0 + s(0)), x + c_1\rangle$ and $\langle c_1 + x, s(0) + x\rangle$ are forks of the TRS $\mathcal{R}_1$ given in Example 1. ◇

Proposition 3. *A flat and linear TRS $\mathcal{R}$ over Σ is UN if and only if for each fork $\langle s, t\rangle$ of $\mathcal{R}$ and every $\mathcal{R}$-normal forms s', t' such that $s \xrightarrow[\mathcal{R}]{*} s'$ and $t \xrightarrow[\mathcal{R}]{*} t'$, it holds $s' = t'$.*

Proof. First, we show that the condition is necessary for unique normalization proceeding by contradiction. Assume that there exists a fork $\langle s, t\rangle$ and two different normal forms s' and t' such that $s \xrightarrow[\mathcal{R}]{*} s'$ and $t \xrightarrow[\mathcal{R}]{*} t'$. It suffices to construct a term u reaching both s and t in order to prove that $\mathcal{R}$ is not uniquely normalizing. Let s and t be of the form $f(s_1, \ldots, s_n)$ and $f(t_1, \ldots, t_n)$, respectively. We construct $u = f(u_1, \ldots, u_n)$ as follows. For every i in $\{1, \ldots, n\}$, if s_i and t_i are the same variable, then we define $u_i := s_i$. Otherwise, if $s_i \xrightarrow[\mathcal{R}]{*} t_i$ then we define $u_i := s_i$. Otherwise, $t_i \xrightarrow[\mathcal{R}]{*} s_i$ holds, and we define $u_i := t_i$. It is clear from this construction that $u \xrightarrow[\mathcal{R}]{*} s$ and $u \xrightarrow[\mathcal{R}]{*} t$ hold, and this concludes the proof for the only if direction.

We prove that the condition is sufficient again by contradiction: we assume that $\mathcal{R}$ is not uniquely normalizing and then prove the existence of a fork not as in Proposition 3. We choose a term u minimal in size with two distinct normal forms v and w reachable from u. The term u cannot be a variable, since variables cannot be rewritten by $\mathcal{R}$. Hence, $u = f(u_1, \ldots, u_n)$ for some $f \in \Sigma_n$ with $n \geq 0$. In order to conclude, it suffices to construct a fork $\langle s, t\rangle$ and two different normal forms s' and t' reachable from s and t, respectively.

For the construction of s, t, s' and t' we proceed iteratively as follows, by initializing them, and modifying them along n steps. The invariant is that s' and t' are always different normal forms reachable from s, and t, respectively, and that every $s|_i$ and $t|_i$ are either both the same variable, or both are $u|_i$, or one of them is a constant reachable from $u|_i$ and the other is $u|_i$, for $i \in \{1, \ldots, n\}$. At the end of the process, $\langle s, t\rangle$ will be a fork.

First, we set $s := u$, $t := u$, $s' := v$ and $t' := w$. After that, for each i in $\{1, \ldots, n\}$, we modify the values of s, t, s' and t' depending on the (un-)definition of $use(i, s \xrightarrow[\mathcal{R}]{*} s')$ and $use(i, t \xrightarrow[\mathcal{R}]{*} t')$.

If $use(i, s \xrightarrow[\mathcal{R}]{*} s')$ is defined to be a constant c, then, by Lemma 1, $s[c]_i \xrightarrow[\mathcal{R}]{*} s'$ and $s|_i \xrightarrow[\mathcal{R}]{*} c$. In this case we just set $s := s[c]_i$ and leave s', t and t' unchanged.

The case where $use(i, s \xrightarrow[\mathcal{R}]{*} s')$ is undefined but $use(i, t \xrightarrow[\mathcal{R}]{*} t')$ is defined to a constant is solved analogously to the previous one.

If both $use(i, s \xrightarrow[\mathcal{R}]{*} s')$ and $use(i, t \xrightarrow[\mathcal{R}]{*} t')$ are undefined, let x be a new variable not occurring in s nor t. By Lemma 2, either $s[x]_i \xrightarrow[\mathcal{R}]{*} s'$, or there exists a position q in $\mathcal{P}os(s')$ such that $s[x]_i \xrightarrow[\mathcal{R}]{*} s'[x]_q$ and $s|_i \xrightarrow[\mathcal{R}]{*} s'|_q$. (This is also analogously true for t and t'.) In any case we set $s := s[x]_i$. In the first case we leave s' unchanged and in the second case we let $s' := s'[x]_q$. We proceed analogously with t and t'. Note that both $s|_i$ and $t|_i$ are now the variable x, and that the new s' and t' are also normal forms reachable from s and t, respectively. But the preservation of the invariant stating that s' and t' are still different requires an explanation. They could only be equal if the same position q has been replaced to x in both terms, and in such a case, the old values $s'|_q$ and $t'|_q$ must be different. This would imply that the old $s|_i$ and $t|_i$ reach different normal forms, and hence, $u|_i$ can reach two different normal forms. But this is in contradiction with the fact that u is a minimal term in size reaching two different normal forms. □

We will use tree automata techniques for checking the condition provided by Proposition 3. But there is a difficulty: with tree automata, we can recognize just ground terms, or terms with variables chosen over a finite set of variables (seen as constants). Fortunately, the condition of Proposition 3 can be simplified by forcing the fork to contain at most two different variables.

Proposition 4. *Let x, y be two distinct variables. A flat and linear TRS $\mathcal{R}$ over Σ is not UN if and only if there exists a fork $\langle s, t\rangle$ of $\mathcal{R}$ with $s, t \in \mathcal{T}(\Sigma, \{x, y\})$ and two different $\mathcal{R}$-normal forms s', t' such that $s \xrightarrow[\mathcal{R}]{*} s'$ and $t \xrightarrow[\mathcal{R}]{*} t'$.*

For proving Proposition 4 it suffices to apply some variable renaming to the given fork by making an adequate use of Lemma 3.

Checking the hypotheses of Proposition 4 requires to test an infinite number of forks. In order to obtain a decision procedure for UN based on the notion of forks, we need a finite representation of such infinite sets of forks. For this purpose, we generalize Definition 2 of forks with regular sets of terms (Definition 3 below), and generalize Proposition 4 into Proposition 5 accordingly. In the next definition, we write $f(L_1, \ldots, L_n)$, where $f \in \Sigma_n$ and $L_1, \ldots, L_n \subseteq \mathcal{T}(\Sigma, X)$ for the set $\{f(t_1, \ldots, t_n) \mid t_1 \in L_1, \ldots, t_n \in L_n\}$.

Definition 3. *Let x, y be two fixed different variables. A* fork of languages *with respect to a flat and linear TRS $\mathcal{R}$ over Σ is a pair $\langle L, L'\rangle$ of sets of terms of $\mathcal{T}(\Sigma, \{x, y\})$ where L and L' have the form $f(L_1, \ldots, L_n)$ and $f(L'_1, \ldots, L'_n)$, respectively, and for every $i \in \{1, \ldots, n\}$, either $L_i = L'_i = \{x\}$, or $L_i = L'_i = \{y\}$, or $L_i = \{c\}$ and $L'_i = (\mathcal{R}^{-1})^*\big(\{c\}\big) \cap \mathcal{T}\big(\Sigma, \{x, y\}\big)$, or $L'_i = \{c\}$ and $L_i = (\mathcal{R}^{-1})^*\big(\{c\}\big) \cap \mathcal{T}\big(\Sigma, \{x, y\}\big)$, for some constant c.*

Recall that $\mathcal{R}^{-1}$ is not necessarily a TRS, but it is an extended TRS. The following lemma is an immediate consequence of Definition 3 (following the assumption that the maximal arity of a function symbol in Σ is fixed).

Lemma 4. *The number of forks of languages with respect to a flat and linear TRS $\mathcal{R}$ is polynomial in the size of $\mathcal{R}$.*

Proof. A fork of languages is determined by choosing a function symbol in Σ_n, and by iterating n times the election of either a constant in Σ_0, placed in one of two component of the fork of languages, or the variable x or the variable y. Thus, there are at most $|\Sigma| \cdot (2|\Sigma|+2)^{max_\Sigma}$ elections. If Σ (and not only max_Σ) is fixed then this is a constant. Otherwise, $|\Sigma|$ can be assumed bounded by $\|\mathcal{R}\|$, and hence, this is a polynomial on $\|\mathcal{R}\|$. □

Proposition 5. *A flat and linear TRS $\mathcal{R}$ over Σ is UN if and only if for all fork of languages $\langle L, L' \rangle$ with respect to $\mathcal{R}$, if $\mathcal{R}^*(L) \cap \mathsf{NF}_\mathcal{R} \neq \emptyset$ and $\mathcal{R}^*(L') \cap \mathsf{NF}_\mathcal{R} \neq \emptyset$, then $\mathcal{R}^*(L) \cap \mathsf{NF}_\mathcal{R} = \mathcal{R}^*(L') \cap \mathsf{NF}_\mathcal{R} = \{t\}$, for some term t.*

Proposition 5 is a direct consequence of Proposition 4.

2.3 Decision of UN

We show now how to decide the condition of Proposition 5, and thus, how to decide UN for flat and linear TRS, using tree automata techniques.

Lemma 5. *Given a flat and linear TRS $\mathcal{R}$ over $\Sigma \cup \{x, y\}$, there exists a TA $\mathcal{A}_\mathcal{R}$ on Σ, of size polynomial in the size of $\mathcal{R}$, recognizing $\mathsf{NF}_\mathcal{R} \cap \mathcal{T}(\Sigma, \{x, y\})$. Moreover, $\mathcal{A}$ can be computed in polynomial time.*

Proof. We construct a TA $\mathcal{A}_\mathcal{R} = (Q, Q^\mathsf{f}, \Delta)$ where $Q = Q^\mathsf{f} = \{q_\alpha \mid \alpha \in ((\Sigma_0 \cup \{x, y\}) \cap \mathsf{NF}_\mathcal{R})\} \cup \{q\}$, and Δ contains one rule $\alpha \to q_\alpha$ for every $q_\alpha \in Q$, and one rule $f(q_1, \ldots, q_n) \to q$ for all $q_1, \ldots, q_n \in Q$ such that the linear term associated to $f(q_1, \ldots, q_n)$ by replacing every occurrence of q_c by c, for each $c \in \Sigma_0$, and all occurrences of q_x, q_y and q by distinct variables, is not reducible by $\mathcal{R}$. The identity $L(\mathcal{A}_\mathcal{R}) = \mathsf{NF}_\mathcal{R} \cap \mathcal{T}(\Sigma, \{x, y\})$ follows by induction on terms for both inclusions. The construction of $\mathcal{A}$ takes time proportional to its size, which is $\mathcal{O}(|\Sigma| \cdot (|\Sigma_0| + 3)^{max_\Sigma})$. □

Note that linearity and flatness are both crucial for the above construction in the given complexity bounds. First, it is known that when $\mathcal{R}$ is not left-linear, then $\mathsf{NF}_\mathcal{R}$ is not necessarily a TA language. Second, $\mathsf{NF}_\mathcal{R} \cap \mathcal{T}(\Sigma, \{x, y\})$ is a TA language as soon as $\mathcal{R}$ is left-linear, but when $\mathcal{R}$ is not flat, there is no polynomial time construction of a TA for $\mathsf{NF}_\mathcal{R}$ with a polynomial number of states. This is a consequence of the EXPTIME lower bound for the ground reducibility of a linear term wrt a linear TRS [3].

The following lemma can be proved using a construction in [13] (Lemma 5.11).

Lemma 6. *Given a TA $\mathcal{A}$ over $\Sigma \cup \{x, y\}$ and an extended flat and linear TRS $\mathcal{R}$ over Σ, there exists a TA of size polynomial in $\|\mathcal{R}\| + \|\mathcal{A}\|$, recognizing $\mathcal{R}^*\big(L(\mathcal{A})\big) \cap \mathcal{T}(\Sigma, \{x, y\})$, which can be constructed in polynomial time.*

There exists a TA construction for larger classes of extended TRS than flat and linear, like right-shallow and right-linear TRS [11]. We focus on the flat and linear case here for complexity reasons. Nevertheless, as we shall see later, UN is undecidable for these larger classes of TRS.

Example 3. The TA $\mathcal{A}'_0 = \left(\{q_0, q_{c_1}\}, \{q_0\}, \Delta\right)$ recognizes $\left(\mathcal{R}_1^{-1}\right)^*(\{0\}) \cap \mathcal{T}(\Sigma, \{x, y\})$, for the TRS $\mathcal{R}_1$ of Example 1, where Δ contains $0 \to q_0$, $c_1 \to q_{c_1}$, $s(q_0) \to q_{c_1}$, $s(q_{c_1}) \to q_0$, $q_0 + q_0 \to q_0$, $q_{c_1} + q_{c_1} \to q_0$, $q_0 + q_{c_1} \to q_{c_1}$ and $q_{c_1} + q_0 \to q_{c_1}$. ◇

Now we shall use the above results for the decision of the sufficient condition for UN given in Proposition 5. The following lemma is a direct consequence of Lemma 6.

Lemma 7. *Let $\mathcal{R}$ be a flat and linear TRS over Σ. For each fork of languages $\langle L, L'\rangle$ with respect to $\mathcal{R}$, L and L' are recognized by two TA over $\Sigma \cup \{x, y\}$ whose respective sizes are polynomial in the size of $\mathcal{R}$, and which can be constructed in polynomial time.*

Now, we have all the ingredients to prove the main result of the paper.

Theorem 1. *UN is decidable in polynomial time for flat and linear TRS.*

Proof. Let $\mathcal{R}$ be a flat and linear TRS over Σ. We construct first a TA $\mathcal{A}_\mathcal{R}$ over $\Sigma \cup \{x, y\}$ recognizing $\mathsf{NF}_\mathcal{R} \cap \mathcal{T}(\Sigma, \{x, y\})$, and whose size $\|\mathcal{A}_\mathcal{R}\|$ is polynomial in the size of $\mathcal{R}$, following Lemma 5.

Now, for each fork of languages $\langle L, L'\rangle$ with respect to $\mathcal{R}$, we perform the following test. Let $\mathcal{A}$ and $\mathcal{A}'$ be two TA of size polynomial in the size of $\mathcal{R}$, recognizing respectively L and L' (constructed according to Lemma 7).

1. Construct two TA recognizing respectively $\mathcal{R}^*(L)$ and $\mathcal{R}^*(L')$. According to Lemma 6, their sizes are polynomial in the size of $\mathcal{R}$.
2. Construct two TA recognizing respectively $\mathcal{R}^*(L) \cap \mathsf{NF}_\mathcal{R}$ and $\mathcal{R}^*(L') \cap \mathsf{NF}_\mathcal{R}$, using the TA constructed at the above step and the above $\mathcal{A}_\mathcal{R}$. According to Proposition 1, the sizes of these two TA are still polynomial in the size of $\mathcal{R}$.
3. If one of the languages $\mathcal{R}^*(L) \cap \mathsf{NF}_\mathcal{R}$ or $\mathcal{R}^*(L') \cap \mathsf{NF}_\mathcal{R}$ is empty (this emptiness test is performed in polynomial time, according to Proposition 2) then the test passes successfully.
4. Otherwise, check whether both languages $\mathcal{R}^*(L) \cap \mathsf{NF}_\mathcal{R}$ and $\mathcal{R}^*(L') \cap \mathsf{NF}_\mathcal{R}$ are singleton sets (this test is performed in polynomial time according to Proposition 2). If it is not the case, the test fails. Otherwise, we construct a TA recognizing their intersection and the test succeeds iff its language is not empty (the construction and emptiness test can still be performed in polynomial time).

According to Proposition 5, $\mathcal{R}$ is UN iff all fork of languages with respect to $\mathcal{R}$ pass the test. According to Lemma 4, there will be a polynomial number of such tests, and following the above evaluation, each test is performed in polynomial time. Altogether, the upper bound on the complexity of deciding UN is polynomial. □

This result can be immediately extended to shallow TRS thanks to the following proposition given in [19].

Proposition 6. *Given a shallow and linear TRS $\mathcal{R}$ over Σ, there exists a flat and linear TRS $\mathcal{R}'$ on an extended signature $\Sigma' \supseteq \Sigma$ such that $\mathcal{R}'$ is UN iff $\mathcal{R}$ is UN. Moreover, the size of $\mathcal{R}'$ is polynomial in the size of $\mathcal{R}$, and can be computed in polynomial time.*

As a consequence of Theorem 1 and Proposition 6 we conclude decidability of UN in polynomial time for shallow and linear TRS.

Theorem 2. *UN is decidable in polynomial time for shallow and linear TRS.*

3 Undecidability of UN for Flat and Right-Linear TRS

Some undecidability results for UN have been recalled in the introduction. In particular, UN is undecidable for right-flat and linear TRS [6]. Thus, the result of Theorem 1 is no longer valid if we relax the assumption on flatness for the left-hand sides of rules. In this section, we show that one can neither relax the assumption on linearity for left-hand sides of rules in Theorem 1. More precisely, we prove (Theorem 3 below) undecidability of UN for flat and right-linear TRS. This result is in contrast with other properties like reachability, joinability, confluence, termination and weak normalization which are all decidable for flat and right-linear TRS [11,8,5,6] (and in some cases for weaker restrictions). The proof involves a reduction from the Post correspondence problem (PCP) restricted to nonempty strings over a fixed finite alphabet Γ.

Problem: restricted-PCP (rPCP).
Instance: a sequence of pairs of words $\langle u_1, v_1\rangle \dots \langle u_n, v_n\rangle$, with $\forall i \leq n$, $u_i, v_i \in \Gamma^* \setminus \epsilon$.
Question: is there a non-empty sequence of indexes $1 \leq i_1, \dots, i_k \leq n$ such that $u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$?

The rPCP is known to be undecidable [14]. We call *solution* of a rPCP instance a non-empty sequence of indexes that gives a positive answer to the above question.

Example 4. The instance of rPCP $\langle ab, a\rangle, \langle c, bc\rangle$ has a solution obtained by choosing the pairs 1 and 2 consecutively, obtaining *abc* at both sides. ◇

In the proof of the undecidability theorem below, we assume a given instance $\langle u_1, v_1\rangle \dots \langle u_n, v_n\rangle$ of rPCP that is, u_i, v_i are nonempty strings over the alphabet Γ. The j'th symbol of u_i and v_i, whenever they exist, are denoted by u_{ij} and v_{ij} respectively. For the sake of readability, we shall sometimes write terms made of symbols of arity 0 and 1 as words over the same symbols.

Theorem 3. *UN is undecidable for flat and right-linear TRS.*

We shall define a TRS $\mathcal{R}$ such that the given rPCP instance has a solution iff $\mathcal{R}$ is not uniquely normalizing. The TRS $\mathcal{R}$ is defined as the union of several flat and right-linear TRS. The role of these sub-TRS is described below, as well as the signature they are based on.

We will represent the words of Γ^* by string terms of $\mathcal{T}\big(\Sigma_A \setminus \{A\}\big)$ where $\Sigma_A := \{\gamma : 1 \mid \gamma \in \Gamma\} \cup \{A : 0, \bot : 0\}$. These terms are generated by a regular tree grammar with a single non-terminal symbol (the constant A), and whose production rules are the rules of the TRS $\mathcal{R}_A$ below.

$$\mathcal{R}_A := \{A \to \gamma(A) \mid \gamma \in \Gamma\} \cup \{A \to \bot\}$$

Let $L = max(|u_1|, \ldots, |u_n|, |v_1|, \ldots, |v_n|)$, and let $\Sigma_U := \{U_{ij} : 1 \mid i \in [1..n], j \in [1..L]\} \cup \{\bot : 0\}$ and $\Sigma_V := \{V_{ij} : 1 \mid i \in [1..n], j \in [1..L]\} \cup \{\bot : 0\}$. Intuitively, we associate to a sequence $i_1, \ldots, i_k$ the pair made of the terms $U_{i_1 1} \ldots U_{i_1 L} \ldots U_{i_k 1} \ldots U_{i_k L} \bot$ and $V_{i_1 1} \ldots V_{i_1 L} \ldots V_{i_k 1} \ldots V_{i_k L} \bot$, which we call a *carrier* of the sequence. The following TRS permits to recover a solution (or more precisely the terms $u_{i_1} \ldots u_{i_k}$ and $v_{i_1} \ldots v_{i_k}$ that must be equal) from a carrier.

$$\begin{aligned}\mathcal{R}_s := \Big\{U_{ij}x \to u_{ij}x \;\Big|\; j \in \big[1..|u_i|\big]\Big\} \cup \Big\{U_{ij}x \to x \;\Big|\; j \in \big[|u_i|+1..L\big]\Big\} \\ \cup \Big\{V_{ij}x \to v_{ij}x \;\Big|\; j \in \big[1..|v_i|\big]\Big\} \cup \Big\{V_{ij}x \to x \;\Big|\; j \in \big[|v_i|+1..L\big]\Big\}\end{aligned}$$

Let us now consider several copies of the terms in carriers of solutions, built over the following signatures:

$$\begin{aligned}\Sigma''_U &:= \big\{U''_{ij} : 1, U'_{ij} : 0 \mid i \in [1..n], j \in [1..L]\big\} \cup \{U : 0, \bot : 0\},\\ \Sigma''_V &:= \big\{V''_{ij} : 1, V'_{ij} : 0 \mid i \in [1..n], j \in [1..L]\big\} \cup \{V : 0, \bot : 0\},\\ \Sigma_P &:= \big\{P_{ij} : 1, P'_{ij} : 0 \mid i \in [1..n], j \in [1..L]\big\} \cup \{P : 0, \bot : 0\}.\end{aligned}$$

The regular grammars generating copies of carriers, using the above (nullary) non-terminal symbols U, V and P, are called respectively R''_U, $\mathcal{R}''_V$ and $\mathcal{R}_P$.

$$\begin{aligned}\mathcal{R}''_U &:= \big\{U \to U'_{i1},\ U'_{ij} \to U''_{ij}U'_{i(j+1)},\ U'_{iL} \to U''_{iL}U,\ U'_{iL} \to U''_{iL}\bot \\ &\qquad\qquad \Big| \; i \in [1..n], j \in [1..L-1]\big\}\\ \mathcal{R}''_V &:= \big\{V \to V'_{i1},\ V'_{ij} \to V''_{ij}V'_{i(j+1)},\ V'_{iL} \to V''_{iL}V,\ V'_{iL} \to V''_{iL}\bot \\ &\qquad\qquad \Big| \; i \in [1..n], j \in [1..L-1]\big\}\\ \mathcal{R}_P &:= \big\{P \to P'_{i1},\ P'_{ij} \to P_{ij}P'_{i(j+1)},\ P'_{iL} \to P_{iL}P,\ P'_{iL} \to P_{iL}\bot \\ &\qquad\qquad \Big| \; i \in [1..n], j \in [1..L-1]\big\}\end{aligned}$$

The following TRS casts carrier of solutions into copies.

$$\mathcal{R}_c := \left\{ \begin{array}{l} U_{ij}x \to U''_{ij}x, U_{ij}x \to P_{ij}x \\ V_{ij}x \to V''_{ij}x,\ V_{ij}x \to P_{ij}x \end{array} \;\middle|\; i \in [1..n], j \in [1..L] \right\}$$

Let us now define the signature

$$\Sigma := \Sigma_A \cup \Sigma_U \cup \Sigma_V \cup \Sigma''_U \cup \Sigma''_V \cup \Sigma_P \cup \{f : 8, 0 : 0, 1 : 0\}$$

The additional symbols f, 0 and 1 are used in the next crux rewrite rules, which act as *checkers* for solutions, in order to ensure the correctness of the reduction.

$$\mathcal{R}_f := \{f(U,V,x,y,x,y,x,y) \to 0,\ f(x,y,A,P,x,x,y,y) \to 1\}$$

Finally, the last TRS $\mathcal{R}_n$ will ensure that 0 and 1 are the only non variable terms in normal form.

$$\begin{aligned}\mathcal{R}_n := &\left\{c \to c \mid c \in \Sigma_0 \setminus \{0,1\}\right\} \cup \{h(x) \to h(x) \mid h \in \Sigma_1\} \\ &\cup \left\{f(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8) \to f(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8)\right\}\end{aligned}$$

To summarize, the TRS $\mathcal{R}$ on Σ is defined by:

$$\mathcal{R} := \mathcal{R}_A \cup \mathcal{R}_s \cup \mathcal{R}''_U \cup \mathcal{R}''_V \cup \mathcal{R}_P \cup \mathcal{R}_c \cup \mathcal{R}_f \cup \mathcal{R}_n.$$

Note that all the rules of $\mathcal{R}$ are flat, all the rules of $\mathcal{R} \setminus \mathcal{R}_f$ are linear, and the rules in $\mathcal{R}_f$ are right-ground. Theorem 3 follows immediately from Lemmas 15 and 8 below, which ensure respectively the correctness and completeness of the reduction of the instance rPCP into the non-UN of $\mathcal{R}$.

Lemma 8. *If the given rPCP instance has a solution then $\mathcal{R}$ is not UN.*

Proof. We assume a solution $i_1, \ldots, i_k$ of the rPCP instance, and show that there exists a term $s \in \mathcal{T}(\Sigma)$ such that $0 \xleftarrow{*}_{\mathcal{R}} s \xrightarrow{*}_{\mathcal{R}} 1$. It permits us to conclude since 0 and 1 are $\mathcal{R}$-normal forms.

Let $w = u_{i_1} \ldots u_{i_k} \bot = v_{i_1} \ldots v_{i_k} \bot$, and let (s_U, s_V) be a carrier of the solution, and s''_U, s''_V, s_P be copies defined as follows.

$$\begin{array}{ll} s_U := U_{i_1 1} \ldots U_{i_1 L} \ldots U_{i_k 1} \ldots U_{i_k L} \bot & s_V := V_{i_1 1} \ldots V_{i_1 L} \ldots V_{i_k 1} \ldots V_{i_k L} \bot \\ s''_U := U''_{i_1 1} \ldots U''_{i_1 L} \ldots U''_{i_k 1} \ldots U''_{i_k L} \bot & s''_V := V''_{i_1 1} \ldots V''_{i_1 L} \ldots V''_{i_k 1} \ldots V''_{i_k L} \bot \\ s_P := P_{i_1 1} \ldots P_{i_1 L} \ldots P_{i_k 1} \ldots P_{i_k L} \bot & s := f(U,V,A,P,s_U,s_U,s_V,s_V) \end{array}$$

It is easy to verify $U \xrightarrow{*}_{\mathcal{R}''_U} s''_U \xleftarrow{*}_{\mathcal{R}_c} s_U$ and $V \xrightarrow{*}_{\mathcal{R}''_V} s''_V \xleftarrow{*}_{\mathcal{R}_c} s_V$. Moreover, A, s_U and s_V rewrite to w using $\mathcal{R}_A$ and $\mathcal{R}_s$. Also, P, s_U and s_V rewrite to s_P using $\mathcal{R}_P$ and $\mathcal{R}_c$. Therefore, there exist derivations $s \xrightarrow{*}_{\mathcal{R}} f(U,V,w,s_P,w,s_P,w,s_P) \xrightarrow[\mathcal{R}_f]{} 0$ and $s \xrightarrow{*}_{\mathcal{R}} f(s''_U,s''_V,A,P,s''_U,s''_U,s''_V,s''_V) \xrightarrow[\mathcal{R}_f]{} 1$ and this concludes the proof. □

Lemma 15 is slightly more difficult and requires some additional definitions and intermediate lemmas for its proof. For space reasons, they are given below without our proof.

Given a word w, we define *indexes*(w) to be the word obtained by applying to w the morphism φ defined as $\varphi(U_{i1}) = \varphi(U''_{i1}) = \varphi(V_{i1}) = \varphi(V''_{i1}) = \varphi(P_{i1}) = i$ and $\varphi(h) = \epsilon$ for any other symbol h. Note that two copies of the same carrier will have the same *indexes*. We say that

a word $w\alpha$ is a *generator* if α is in $\{U, V, A, P, U'_{ij}, V'_{ij}, P'_{ij} | i \in [1..n], j \in [1..L]\}$. Otherwise, if α is $\bot$, we say that $w\alpha$ is a *non-generator*. We define Σ_c as the subset of Σ_1 of the unary symbols h for which there exists a collapsing rule $h(x) \to x$ in $\mathcal{R}$, i.e. Σ_c contains all U_{ij} such that $j > |u_i|$, and all V_{ij} such that $j > |v_i|$. For any term t define $clean(t)$ recursively as follows. If the top symbol h of t is in Σ_c then we define $clean(t) = clean(t|_1)$. Otherwise, we define $clean(t) = t$. In other words, $clean(w\alpha)$ removes from t the longest prefix of unary symbols with all them in Σ_c.

Lemma 9. *Let s and t be terms satisfying $s \xrightarrow[\mathcal{R}]{*} t$. Then $clean(s) \xrightarrow[\mathcal{R}]{*} clean(t)$.*

Lemma 10. *If $s \xrightarrow[\mathcal{R}]{*} U$ then $clean(s) = U$; if $s \xrightarrow[\mathcal{R}]{*} V$ then $clean(s) = V$; if $s \xrightarrow[\mathcal{R}]{*} A$ then $clean(s) = A$; if $s \xrightarrow[\mathcal{R}]{*} P$ then $clean(s) = P$.*

Lemma 11. *A word $w\alpha$ is necessarily a non-generator if both $\{w\alpha, U\}$ and $\{w\alpha, A\}$, or both $\{w\alpha, V\}$ and $\{w\alpha, A\}$, or both $\{w\alpha, U\}$ and $\{w\alpha, P\}$, or both $\{w\alpha, V\}$ and $\{w\alpha, P\}$ are joinable.*

Lemma 12. *Let $w_1\bot$ and $w_2\bot$ be two non-generator words. Let α be either U or V or P. If $\{w_1\bot, w_2\bot, \alpha\}$ is $\mathcal{R}$-joinable, then $indexes(w_1) = indexes(w_2)$.*

Lemma 13. *Let $w_1\bot$ be a non-generator word joinable with U. Let $w_2\bot$ be a word reachable from $w_1\bot$ and such that $w_2 \in \Gamma^*$. If $i_1, \ldots, i_k = indexes(w_1)$ then $w_2 = u_{i_1} \ldots u_{i_k}$.*

The following lemma is the analogous to the previous lemma, but with V's and v's instead of U's and u's.

Lemma 14. *Let $w_1\bot$ be a non-generator word joinable with V. Let $w_2\bot$ be a word reachable from $w_1\bot$ and such that $w_2 \in \Gamma^*$. If $i_1, \ldots, i_k = indexes(w_1)$ then $w_2 = v_{i_1} \ldots v_{i_k}$.*

Now, we have all the necessary ingredients for proving Lemma 15.

Lemma 15. *If $\mathcal{R}$ is not UN then the given rPCP instance has a solution.*

Proof. Let s be a term in $\mathcal{T}(\Sigma, \mathcal{V})$, minimal in size, such that $s' \xleftarrow[\mathcal{R}]{*} s \xrightarrow[\mathcal{R}]{*} s''$ where s' and s'' are two distinct $\mathcal{R}$-normal forms. Note that only 0, 1 and variables are $\mathcal{R}$-normal forms.

The term s is not rooted by a symbol in $\Sigma \setminus (\Sigma_c \cup \{f, 0, 1\})$: otherwise it could not reach a $\mathcal{R}$-normal form, because of the rules of $\mathcal{R}_n$ and because such symbols cannot be removed.

Assume that s is rooted by a symbol h in Σ_c. Thus, s is of the form $h(s_1)$ for some term s_1. Then in both derivations $s \xrightarrow[\mathcal{R}]{*} s'$ and $s \xrightarrow[\mathcal{R}]{*} s''$, the rule $h(x) \to x$ is applied at the root position. Hence, the term s_1 also reaches s' and s'' with $\mathcal{R}$, contradicting the minimality of s.

Assume that s is rooted by 0 or 1 or a variable. It means that s is directly 0 or 1 or a variable. Any of these terms is a normal form, and this is in contradiction with the fact that s reaches two different $\mathcal{R}$-normal forms.

From the above observations it follows that s is of the form $f(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8)$ for some terms $s_1, \ldots, s_8$. Since no variable is reachable from a term of this form (there is no collapsing rule in $\mathcal{R}$ with a f at the top of its left-hand side), we conclude that $s' = 0$ and $s'' = 1$, or vice-versa. Thus, the two rules of $\mathcal{R}_f$ are applied at the root position in the derivations from s and we have the following rewrite sequences:

$$0 \xleftarrow[\mathcal{R}_f]{} f(U, V, s_3^0, s_4^0, s_3^0, s_4^0, s_3^0, s_4^0) \overset{*}{\underset{\mathcal{R}\backslash\mathcal{R}_f}{\leftarrow}} f(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8) \overset{*}{\underset{\mathcal{R}\backslash\mathcal{R}_f}{\rightarrow}} f(s_1^1, s_2^1, A, P, s_1^1, s_1^1, s_2^1, s_2^1) \xrightarrow[\mathcal{R}_f]{} 1$$

Note that only rewrite steps with $f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) \rightarrow f(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ occur at the root position in the two above diagonal rewrite sequences. This implies that each subterm at depth 1 in $f(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8)$ reaches the subterm at depth 1 in $f(U, V, s_3^0, s_4^0, s_3^0, s_4^0, s_3^0, s_4^0)$ located at the same position, and also the subterm at depth 1 in $f(s_1^1, s_2^1, A, P, s_1^1, s_1^1, s_2^1, s_2^1)$ located at the same position.

By Lemma 9, we can assume without loss of generality that any subterm t at depth 1 in any of those three terms satisfies $\mathit{clean}(t) = t$. Moreover, by Lemma 10, it follows $s_1 = U$, $s_2 = V$, $s_3 = A$ and $s_4 = P$. This implies the following facts.

- U and s_5 are joinable, and s_5 and A are joinable, and hence, by Lemma 11, s_5 is a non-generator. Similarly, s_6, s_7 and s_8 are non-generators.
- $\{U, s_5, s_6\}$ is joinable, and hence, by Lemma 12, $\mathit{indexes}(s_5) = \mathit{indexes}(s_6)$. Similarly, $\mathit{indexes}(s_7) = \mathit{indexes}(s_8)$, and $\mathit{indexes}(s_6) = \mathit{indexes}(s_8)$. Let $i_1, \ldots, i_k$ be *indexes* of any of them.
- $\{U, s_5\}$ is joinable, and $\{A, s_5\}$ is joinable to a word s_3^0 of the form $w\bot$ such that $w \in \Gamma^*$. Hence, by Lemma 13, $w = u_{i_1} \ldots u_{i_k}$. Similarly, $\{V, s_7\}$ is joinable, and $\{A, s_7\}$ is joinable to the same $s_3^0 = w\bot$. Hence, by Lemma 14, $w = v_{i_1} \ldots v_{i_k}$. Thus, $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$.

From the above facts, it follows that $i_1, \ldots, i_k$ is a solution of the given rPCP instance. □

Conclusion

We have shown that UN is decidable in polynomial time for shallow and linear TRS (Theorem 2), and is undecidable for flat and right-linear TRS (Theorem 3). With these results, the problem of decidability of UN for classes of TRS defined by syntactic restrictions like linearity, flatness or shallowness is essentially closed. Perhaps, one could still consider this problem for other variants of syntactic restrictions based on the form of the dependency pairs obtained from a TRS, like the ones given in [20].

References

1. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press, New York (1998)
2. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007), `http://tata.gforge.inria.fr`
3. Comon, H., Jacquemard, F.: Ground reducibility is EXPTIME-complete. Information and Computation 187(1), 123–153 (2003)
4. Godoy, G., Hernández, H.: Undecidable properties for flat term rewrite systems (submitted) (2008)
5. Godoy, G., Huntingford, E., Tiwari, A.: Termination of rewriting with right-flat rules. In: Baader, F. (ed.) RTA 2007. LNCS, vol. 4533, pp. 200–213. Springer, Heidelberg (2007)
6. Godoy, G., Tison, S.: On the normalization and unique normalization properties of term rewrite systems. In: Pfenning, F. (ed.) CADE 2007. LNCS, vol. 4603, pp. 247–262. Springer, Heidelberg (2007)
7. Godoy, G., Tiwari, A.: Deciding fundamental properties of right-(ground or variable) rewrite systems by rewrite closure. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004. LNCS (LNAI), vol. 3097, pp. 91–106. Springer, Heidelberg (2004)
8. Godoy, G., Tiwari, A.: Confluence of shallow right-linear rewrite systems. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 541–556. Springer, Heidelberg (2005)
9. Jacquemard, F.: Reachability and confluence are undecidable for flat term rewriting systems. Inf. Process. Lett. 87(5), 265–270 (2003)
10. Mitsuhashi, I., Oyamaguchi, M., Jacquemard, F.: The confluence problem for flat TRSs. In: Calmet, J., Ida, T., Wang, D. (eds.) AISC 2006. LNCS (LNAI), vol. 4120, pp. 68–81. Springer, Heidelberg (2006)
11. Nagaya, T., Toyama, Y.: Decidability for left-linear growing term rewriting systems. Inf. Comput. 178(2), 499–514 (2002)
12. Nieuwenhuis, R.: Basic paramodulation and decidable theories (extended abstract). In: LICS, pp. 473–482 (1996)
13. Salomaa, K.: Deterministic tree pushdown automata and monadic tree rewriting systems. J. Comput. Syst. Sci. 37, 367–394 (1998)
14. Sipser, M.: Introduction to the Theory of Computation. Course Technology (2006)
15. Takai, T., Kaji, Y., Seki, H.: Right-linear finite path overlapping term rewriting systems effectively preserve recognizability. In: Bachmair, L. (ed.) RTA 2000. LNCS, vol. 1833, pp. 246–260. Springer, Heidelberg (2000)
16. Verma, R.: Complexity of normal form properties and reductions for rewriting problems. Fundamenta Informaticae (accepted for publication) (2008)
17. Verma, R.: New undecidability results for properies of term rewrite systems. In: 9th International workshop, RULE, pp. 1–16 (2008)
18. Verma, R., Hayrapetyan, A.: A new decidability technique for ground term rewriting systems. ACM Trans. Comput. Log. 6(1), 102–123 (2005)
19. Verma, R., Zinn, J.: A polynomial-time algorithm for uniqueness of normal forms of linear shallow term rewrite systems. In: Symposium on Logic in Computer Science LICS (short presentation) (2006)
20. Wang, Y., Sakai, M.: Decidability of termination for semi-constructor trss, left-linear shallow TRSs and related systems. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 343–356. Springer, Heidelberg (2006)

The Existential Fragment of the One-Step Parallel Rewriting Theory

Aleksy Schubert*

Institute of Informatics
Warsaw University
ul. Banacha 2
02–097 Warsaw
Poland

SoS Group
NIII
Faculty of Science
University of Nijmegen
The Netherlands

Abstract. It is known that the first-order theory with a single predicate $\rightarrow$ that denotes a one-step rewriting reduction on terms is undecidable already for formulae with $\exists\forall$ prefix. Several decidability results exist for the fragment of the theory in which the formulae start with the $\exists$ prefix only. This paper considers a similar fragment for a predicate $\rightarrow^p$ which denotes the parallel one-step rewriting reduction. We show that the first-order theory of $\rightarrow^p$ is undecidable already for formulae with $\exists^7$ prefix and left-linear rewrite systems.

1 Introduction

The first-order one-step parallel rewriting theory is a first-order theory which has only one relation symbol $\rightarrow^p$. The logical value of formulae in this theory is checked in a structure of ground terms over a signature Σ. Two terms s, t are in the relation $\rightarrow^p$ when s can be rewritten in one parallel step to t using rewrite rules from a fixed finite set R. It is worth noting that the formulae of the theory cannot use the equality relation $=$ and function symbols from Σ. There is also no direct way to express the fact that a particular rewriting is done with a specific set of rules from R.

The notion of the parallel term rewriting emerged in the studies on computational frameworks [GKM87]. This model of computation supports the study of computations in the context of concurrent or parallel programming [AK96]. Efficient implementations of parallel rewriting systems rely on various graph rewriting techniques which were intensely studied (for an overview see e.g. [Ass00]). Parallel rewriting has also been used as a basis for the logical framework of rewriting logic [MOM02] as well as in the context of regular tree languages [STT97].

The first-order theory of one-step rewriting, but in the non-parallel case, has been proved to be undecidable by Treinen [Tre96, Tre98]. This result was further strengthened to work for various weak classes of rewriting systems: linear, shallow

* This work was partly supported by KBN grant 3 T11C 002 27 and Sixth Framework Programme MEIF-CT-2005-024306 SOJOURN.

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 78–92, 2009.
© Springer-Verlag Berlin Heidelberg 2009

[STT97, STTT01]; linear, terminating [Vor97, Mar97]; right-ground, terminating [Mar97]; and finitely-terminating, linear, confluent [Vor02]. The proofs in (most of) the papers above showed undecidability of $\exists^*\forall^*$ fragment of the theory. The strongest of them was [Tre96] where already $\exists^2\forall$ fragment is undecidable. However, this was not obtained for a fixed rewrite system. Vorobyov in [Vor02] showed a fixed system for which $\exists\forall^3$ fragment is undecidable.

Despite the negative results in the general case, researchers investigated a special case in which the existential formulae of the one-step rewriting theory were considered. The currently existing results show decidability of certain subclasses of the theory. The early theorems in [Tis90] imply that the theory is decidable in the positive case (i.e. for formulae with no negation) in the case of left-linear right-ground systems. More generally, the existential fragment with the positive formulae is decidable for arbitrary rewriting systems [NPR97]. The case of formulae with negation lead to a more specific consideration of the problem with regard to the classes of rewriting rules and resulted in the decidability for quasi-shallow rewriting systems [CSTT99]; linear, non left-left-overlapping; and non ϵ-left-right-overlapping systems [LR99]. Interestingly enough, the whole one-step rewriting logic is decidable for unary signatures [Jac96].

This paper is organised in the following way. We fix the notation in Section 2. Subsequently, we present the undecidability proof in Section 3. The presentation of the proof is divided into two subsections. The first of them (3.1) presents a slightly modified version of the Turing machine, which is easier to handle in the proof, and the second (3.2) presents a class of rewriting systems that simulates the work of the machine which leads to a proof of undecidability of the rewriting theory we deal with here. We conclude the paper with a discussion in Section 4.

2 Preliminaries

This section recalls preliminary notions used in the rest of the paper and fixes the notation.

The function symbols belong to signatures which are usually denoted by Σ, Σ', etc. Each symbol has its arity. The symbols of non-zero arity in Σ are usually denoted by letters such as f, g etc. The zero arity symbols are denoted by c, d etc. Let X be disjoint with Σ. The symbols from X are used as variables and can be considered to be symbols of arity 0 when forming terms with variables. The variables are, by convention, written x, y etc. When convenient, we do not distinguish between terms, and trees labelled with the symbols from Σ or $\Sigma \cup X$. The set of all finite ground terms over a signature Σ is denoted by $T(\Sigma)$. The set of terms with variables in X is denoted by $T(\Sigma, X)$. The terms are usually denoted by small Latin letters such as t, s, u, etc. The set of variables that occur in a term t is denoted by $\mathrm{FV}(t)$. A substitution can replace occurrences of variables in a term t with some other terms. Substitutions are usually noted as S, T etc. and the result of the application of a substitution S to a term t is written as $S(t)$. We denote by $C[-_1, \ldots, -_n]$ a context which contains $n > 0$ placeholders (each occurring exactly once). The placeholders may be replaced by particular terms $t_1, \ldots, t_n$ which is denoted as $C[t_1, \ldots, t_n]$.

In order to address positions in a tree we use sequences of natural numbers. The addresses are denoted by small Greek letters such as γ, ρ, etc. The root address (empty sequence) is denoted by ε. The subterm (subtree) of t at an address γ is denoted by $t|_\gamma$. We compose addresses so that $f(t_1, t_2)|_{i \cdot \gamma} = t_i|_\gamma$ for $i = 1, 2$. The result of the replacement of the subtree at an address γ in t with a tree s is denoted by $t[\gamma \leftarrow s]$.

Let R be a finite set of pairs of terms $\langle l, r \rangle$ over a signature Σ such that $\mathrm{FV}(r) \subseteq \mathrm{FV}(l)$. We call such pairs *rewrite rules* over Σ and write them $l \to r$. We say that a term t *rewrites* to a term s with a rule $l \to r$ when there is an address γ in both t and s and a substitution S such that $t = t[\gamma \leftarrow S(l)]$ and $s = t[\gamma \leftarrow S(r)]$. We say that a term t *rewrites* to a term s (written $s \to t$) when t rewrites to s with some rule $l \to r \in R$. We say that R is a *left-linear* rewrite system in case each variable occurs in l at most once.

2.1 One-Step Parallel Rewriting Theory

This subsection presents the notions concerning the first-order parallel one-step rewriting theory. We fix a signature Σ which contains at least one symbol of arity 0.

Definition 1. (definition of $\to^p$)
Let R be a set of rewrite rules over Σ. We consider a relational structure $\mathcal{A}_R = \langle T(\Sigma), \to^p \rangle$ where the symbol $\to^p$ represents the one-step parallel rewriting and is defined as follows: $t \to^p {}^{\mathcal{A}} s$ for $t, s \in T(\Sigma)$ iff there is a context $C[-_1, \ldots, -_k]$ with $k > 0$ such that $t = C[t_1, \ldots, t_k]$ and $s = C[s_1, \ldots, s_k]$ and for $i = 1, \ldots, k$ we have $t_i \to s_i$. Additionally, we use the symbol $\to^p_*$ to denote the reflexive-transitive closure of $\to^p$.

Note that the domain of the structure consists of the terms with no variables. Therefore, the signatures must be restricted to contain at least one symbol of arity 0.

The atomic formulae of the first-order one-step parallel rewriting theory are of the shape $x \to^p y$ only (no formulae of the form $x = y$). These atomic formulae can be combined with $\neg, \wedge$ and $\vee$. Free variables can be bound by quantifiers $\exists, \forall$. We write $x \not\to^p y$ as a shorthand for $\neg x \to^p y$.

It is worth pointing out that the context C used in the definition above has at least one placeholder. This design choice makes the notion closer to the notion of the usual one-step rewriting as in both cases at least one rewrite rule is executed.

The existential fragment of the first-order one-step parallel rewriting theory consists of closed formulae of the form $\exists x_1 \cdots \exists x_n . \phi$ where ϕ does not contain quantifiers.

The theory of a parallel one-step rewriting is neither stronger nor weaker than the non-parallel one.

Proposition 1. *There is a signature Σ and a term rewriting system R such that:*

(1) There is an existential formula ϕ_1, with $\rightarrow$ as the only predicate, such that the formula holds in case $\rightarrow$ is interpreted as the one-step rewriting according to R and does not hold in case $\rightarrow$ is interpreted as the parallel one-step rewriting.

(2) There is an existential formula ϕ_2, with $\rightarrow$ as the only predicate, such that the formula holds in case $\rightarrow$ is interpreted as the parallel one-step rewriting according to R and does not hold in case $\rightarrow$ is interpreted as the one-step rewriting.

Proof. Consider the signature $\Sigma = \{f, a, b\}$ where f is binary and a, b are constants and a rewrite system R with a rule $a \rightarrow b$.

For the proof of (1) consider a formula $\exists xyz.x \rightarrow y \wedge y \rightarrow z \wedge \neg x \rightarrow z$. For the proof of (2) consider a formula $\exists xy.x \rightarrow y \wedge x \rightarrow z \wedge z \rightarrow y$. The details are left to the reader.

We can also fix a formula and manipulate the rewrite systems:

Proposition 2. *There is a signature Σ and a formula ϕ such that:*

(1) There is a rewrite system R_1 such that ϕ holds in case $\rightarrow$ is interpreted as the one-step rewriting according to R_1 and does not hold in case $\rightarrow$ is interpreted as the parallel one-step rewriting.

(2) There is a rewrite system R_2 such that ϕ holds in case $\rightarrow$ is interpreted as the parallel one-step rewriting according to R_2 and does not hold in case $\rightarrow$ is interpreted as the non-parallel one-step rewriting.

Proof. Consider the signature $\Sigma = \{f, a, b, c\}$ where f is binary and a, b, c are constants and a formula

$$\phi = \exists x_1 x_2 x_3.\ x_1 \rightarrow x_1 \wedge x_1 \rightarrow x_2 \wedge x_1 \rightarrow x_3 \wedge \\ \neg x_2 \rightarrow x_1 \wedge \neg x_2 \rightarrow x_2 \wedge x_2 \rightarrow x_3 \wedge x_3 \rightarrow x_3.$$

For the proof of (1) consider a rewrite system $R_1 = \{b \rightarrow a\} \cup \{f(a, a) \rightarrow f(t, t) \mid t = a, b, c\} \cup \{f(t, t) \rightarrow f(c, c) \mid t = a, b, c\}$. For the proof of (2) consider a rewrite system $R_2 = \{b \rightarrow c, f(a, b) \rightarrow f(a, b), f(a, b) \rightarrow f(b, b), f(a, b) \rightarrow f(c, c), f(c, c) \rightarrow f(c, c)\}$.

A deeper relation between the theories requires further investigation. In particular, we conjecture that

Conjecture 1. There is an effective translation Φ_1 which transforms a rewrite system R_1 into a rewrite system $\Phi_1(R_1)$ and an effective translation Ψ_1 on first order formulae such that for each formula ϕ: ϕ holds when $\rightarrow$ is interpreted as one-step rewriting according to R_1 iff $\Psi_1(\phi)$ holds when $\rightarrow$ is interpreted as one-step parallel rewriting according to $\Phi_1(R_1)$.

It is straightforward to prove this conjecture in a more expressive setting in which we are able to use all the relations respectively $\rightarrow_r$ or $\rightarrow_r^p$ where r is a rewrite rule from the rewrite system in question. The construction is, however, not as obvious in the setting we deal with here.

The main decision problem we deal with in this paper is the following:

Definition 2. `(satisfiability problem)`
Input: $\langle \Sigma, \phi, R \rangle$ where ϕ is a formula of the first-order one-step parallel rewriting theory and R is a rewrite system over Σ (Σ contains at least one constant).
Question: Is ϕ satisfied in the structure $\mathcal{A}_R$ over the signature Σ?

It is worth pointing out that the construction of Treinen [Tre98] can immediately be adapted to prove the undecidability of the problem in case of $\exists^2\forall^*$ formulae.

3 The Undecidability Construction

We reduce here the halting problem for a special kind of Turing machines — left-terminal Turing machines — to the validity of existential formulae in the first-order one-step parallel rewriting logic. Thus we obtain our undecidability.

3.1 Left-Terminal Turing Machines

Let $M = \langle Q, q_I, Q_f, \Sigma, \rightarrow_M \rangle$ be a deterministic Turing machine (DTM), where Q is a set of its states, q_I is the initial state, Q_f is a set of its final states, Σ is an alphabet of the Turing machine, and $\rightarrow_M \colon Q \times \Sigma \rightharpoonup Q \times \Sigma \times D$ is a partial next step function with D being the set of directions $\{L, R, S\}$. We usually write $q, a \rightarrow_M q', a', d$ instead of $\rightarrow_M (q, a) = (q', a', d)$. We assume a model in which the Turing machine tape extends on demand with a cell that contains 0 when the machine tries to move rightwards while there is no next tape cell.

Definition 3. `(left-terminal Turing machine)`
We say that DTM M is a *left-terminal Turing machine* (LTTM) when

1. $\Sigma = \{0, 1\}$,
2. for each q such that $q, w \rightarrow_M q', l, d$ we have $d \neq S$,
3. the final configuration is reached only at the left end of the tape.

The goal of the restriction (1) is to downsize the number of rewrite rules. Thanks to the restriction (2), we avoid the technical difficulties caused by moves that do not change the machine tape. The restriction (3) serves to enable a possibility to check by rewrite rules that the terminal configuration is reached.

A *configuration* of an LTTM M is a sequence $\rho_1 \cdot (q, l) \cdot \rho_2$ where $\rho_1, \rho_2 \in \Sigma^*, l \in \Sigma$, and $q \in Q$. The *initial configuration* of M is a configuration of the form $(q_I, l) \cdot \gamma$ where q_I is the initial state of M and $l \cdot \gamma$ is the input sequence for the Turing machine. The relation $\rightarrow_M$ extends as usual so that it relates pairs of configurations.

The *equivalence* $\sim_M$ is the symmetric, reflexive and transitive closure of the $\rightarrow_M$ relation on configurations; the reflexive and transitive closure of $\rightarrow_M$ on configurations is denoted $\rightarrow_M^*$. Let γ_1 be an initial configuration and γ_n a final configuration. A sequence $\gamma_1, \ldots, \gamma_n$ of configurations that is a witness of $\gamma_1 \sim_M \gamma_n$ equivalence is called an *eq-run*. We consider the following equivalence problem

Definition 4. (the equivalence problem)
Input: An initial configuration $(q_I, l) \cdot \rho$ of an LTTM M.
Question: Is there a final configuration $(q, l') \cdot \rho'$ where $q \in Q_f$, and $l' \cdot \rho' \in \Sigma^*$, such that $(q_I, l) \cdot \rho \sim_M (q, l') \cdot \rho'$?

We deal with this problem rather than with the reachability problem as it allows us to express that certain term reductions (e.g. $t_3 \to^p t_{32}$ for terms in Def. 10) are not caused by a move of an LTTM we simulate. The following theorem holds:

Theorem 1. (the undecidability of the equivalence)
The equivalence problem for LTTM is undecidable.

Proof. The proof is by a routine technique similar to the proof of undecidability of Thue systems.

3.2 Rewriting and LTTM

We relate here the term rewriting and the existential fragment of the parallel rewriting logic.

Definition 5. (existential formula)
Let

$$\begin{aligned}
\phi_{\text{form}} &= x_1 \not\to^p x_1 \\
\phi_{\text{start}} &= x_1 \to^p x_3 \wedge x_3 \not\to^p x_1 \wedge x_3 \to^p x_3 \wedge \phi_{\text{loop}}(3) \\
\phi_{\text{run}} &= x_1 \not\to^p x_2 \wedge x_2 \to^p x_1 \wedge x_2 \to^p x_2 \wedge \phi_{\text{loop}}(2) \\
\phi_{\text{end}} &= x_3 \to^p x_2 \wedge x_2 \not\to^p x_3 \\
\phi_{\text{loop}}(i) &= x_i \to^p x_{i1} \wedge x_{i1} \to^p x_i \wedge x_{i1} \to^p x_{i2} \wedge x_{i2} \to^p x_i \wedge x_i \not\to^p x_{i2}
\end{aligned}$$

The resulting existential sentence is:

$$\phi_M = \exists x_1 x_2 x_3 x_{21} x_{22} x_{31} x_{32}.\phi_{\text{form}} \wedge \phi_{\text{start}} \wedge \phi_{\text{run}} \wedge \phi_{\text{end}}.$$

To save the notational burden the parameter of the formula ϕ_{loop} is an index. The formula ϕ_{form} guarantees that the terms we deal with here are appropriate candidates for encodings of an eq-run. The formula ϕ_{start} serves to ensure that a simulation of the LTTM starts with a starting configuration of the machine, ϕ_{end} to ensure that the simulation ends with a final configuration of the machine, and at last ϕ_{run} serves to guarantee that the simulation will obey the transition rules of the machine. The formulae $\phi_{\text{loop}}(2)$ and $\phi_{\text{loop}}(3)$ ensure that the terms substituted for x_2 and x_3 contain special syntactical pattern possible only after the starting configuration is checked. The dependencies between the variables $x_1, x_2, x_3, x_{21}, x_{22}, x_{31}, \ldots, x_{32}$ encoded by ϕ_M are presented on Fig. 1.

Let us fix an LTTM M. Based on that we construct a signature Σ_M, a rewriting system R_M and a sentence ϕ_M such that M halts iff ϕ_M holds in $\mathcal{A}_{R_M}$.

Definition 6. (signature)
Now, we define the signature Σ_M of the rewriting system which corresponds to a LTTM M. It contains the following symbols

$$f, g \text{ of arity } 2, \qquad \perp_g, \perp_g^1, \perp_g^2, \perp_g^3 \text{ of arity } 1, \qquad \perp_f, 0, 1 \text{ of arity } 0 \quad .$$

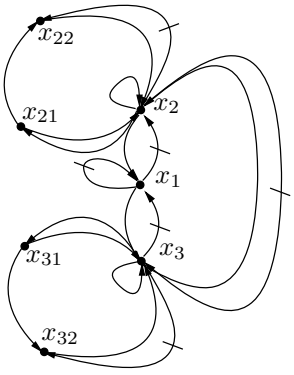


Fig. 1. The relationships between variables in the existential formula. The nodes represent corresponding variables, the normal edges ($\rightarrow$) represent the relation $\rightarrow^p$, the crossed edges ($\not\rightarrow$) represent the negation of the relation $\rightarrow^p$.

Additionally, we use elements of the set $Q \times \{0, 1\}$ as symbols of arity 0 where Q is the set of states of M.

Note that the restriction (1) from Def. 3 makes possible to use here the symbols $0, 1$ only. The intent of the formula ϕ_M is to enforce that the term t_1 substituted for x_1 contains an encoded eq-run of the LTTM we are interested in. Furthermore, it ensures that the term t_3 substituted for x_3 will be almost the same with one exception that $\bot_g$ is replaced with $\bot_g^1$. The formula $\phi_{\text{loop}}(3)$ guarantees roughly that $\bot_g^1$ actually occurs there. The term t_2 substituted for x_2 also contains the symbol which is guaranteed by $\phi_{\text{loop}}(2)$. The loop between x_1, x_3 and x_2 guarantees that t_2 differs from t_3 so that the terminal configuration at the top of t_3 is deleted from t_2. In this way, we obtain the opportunity to compare subsequent machine moves in the fashion presented on Fig. 2.

Now, we are able to define an encoding of a configuration and an eq-run.

Definition 7. `(encoding of configurations)`
Let ρ be a configuration of M. We define the encoding $\text{enc}(\rho)$ of the configuration in the term algebra over the signature from Def. 6 as follows:

- $\text{enc}(\varepsilon) = \bot_f$;
- $\text{enc}(l \cdot \rho') = f(l, \text{enc}(\rho'))$ for $l \in \{1, 0\}$;
- $\text{enc}((q, l) \cdot \rho') = f(\langle q, l\rangle, \text{enc}(\rho'))$ where $l \in \{1, 0\}$.

Definition 8. `(encoding of an eq-run)`
Let $\rho_1, \ldots, \rho_n$ be an eq-run of M. The term $\text{e2t}(\rho_1, \ldots, \rho_n)$ is defined as:

- $\text{e2t}(\rho_1) = g(\bot_g(t), t)$, where $t = \text{enc}(\rho_1)$;
- $\text{e2t}(\rho_1, \ldots, \rho_i) = g(\text{e2t}(\rho_1 \ldots, \rho_{i-1}), \text{enc}(\rho_i))$.

Given an LTTM M and its initial state q_I together with the input word $l \cdot \rho_1$, we can define a rewriting system which has the behaviour equivalent to the behaviour of $\sim_M$ on $l \cdot \rho_1$.

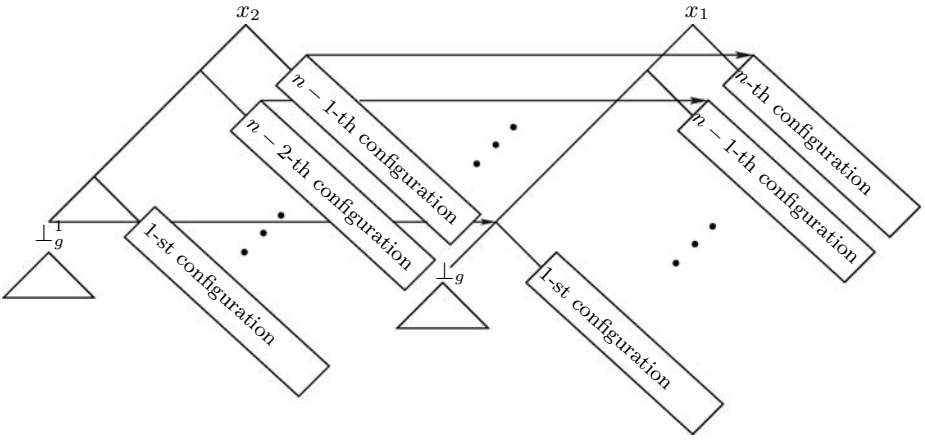


Fig. 2. The mechanism for the next step simulation. The horizontal arrows relate subterms at the same path in terms substituted for x_2 and x_1.

Definition 9. `(rewrite system)`
The rewrite rules can be divided as follows:

1. The rules relevant to the detection of the starting configuration:

$$[1]\ \perp_g(\mathrm{enc}((q_I, l)\cdot\rho_1)) \to \perp_g^1(\mathrm{enc}((q_I, l)\cdot\rho_1)),$$
$$[2]\ \perp_g^1(x) \to \perp_g^1(x), \qquad [3]\ \perp_g^1(x) \to g(\perp_g(x), x)$$

   where $(q_I, l)\cdot\rho_1$, as above, is the initial configuration of M.
2. The rules relevant to detection of the final configuration:

$$[1]\ g(g(x,y), f(\langle q, l\rangle, z)) \to g(x,y)$$

   where l ranges over $\{0,1\}$, and $q \in Q_f$ (Q_f contains the final states of M).
3. The rules that check the form of an eq-run:

$$[1]\ f(f(x,y),z) \to f(f(x,y),z), \qquad [2]\ g(x,g(y,z)) \to g(x,g(y,z)),$$
$$[3]\ f(g(x,y),z) \to f(g(x,y),z), \qquad [4]\ f(x,g(y,z)) \to f(x,g(y,z)),$$
$$[5]\ g(f(x,y),z) \to g(f(x,y),z), \qquad [6]\ g(x,\perp_g(y)) \to g(x,\perp_g(y)),$$
$$[7]\ f(\perp_g(x),y) \to f(\perp_g(x),y), \qquad [8]\ f(x,\perp_g(y)) \to f(x,\perp_g(y)),$$
$$[9]\ g(c_1,x) \to g(c_1,x), \qquad [10]\ g(x,c_2) \to g(x,c_2),$$
$$[11]\ \perp_g(\perp_g(x)) \to \perp_g(\perp_g(x)), \qquad [12]\ \perp_g(g(x,y)) \to \perp_g(g(x,y)),$$
$$[13]\ \perp_g(c_2) \to \perp_g(c_2), \qquad [14]\ f(x,c_2) \to f(x,c_2),$$
$$[15]\ f(\perp_f,x) \to f(\perp_f,x), \qquad [16]\ \perp_g^i(x) \to \perp_g^i(x) \quad \text{for } i=1,2,3,$$
$$[17]\ g(g(x,f(\langle q,l\rangle,y)), f(\langle q,l\rangle,z)) \to g(g(x,f(\langle q,l\rangle,y)), f(\langle q,l\rangle,z))$$

   where c_1 ranges over all symbols of arity 0, c_2 over all symbols of arity 0 except from $\perp_f$, l ranges over $\{0,1\}$, and q ranges over Q_f.

4. The rules which check that an eq-run obeys the state transitions: all the rules of the form
$$\begin{array}{l}[1]\ f(\langle q_1, l_1\rangle, f(l_2, x)) \to f(l_3, f(\langle q_2, l_2\rangle, x)),\\ [2]\ f(\langle q_1, l_1\rangle, \bot_f) \to f(l_3, f(\langle q_2, 0\rangle, \bot_f))\end{array} \tag{1}$$
in case $q_1, l_1 \to_M q_2, l_3, R$ where $q_1, q_2 \in Q$, $l_1, l_2, l_3 \in \{0, 1\}$; and rules of the form
$$[3]\ f(l_1, f(\langle q_1, l_2\rangle, x)) \to f(\langle q_2, l_1\rangle, f(l_3, x)), \tag{2}$$
in case $q_1, l_2 \to_M q_2, l_3, L$ where $q_1, q_2 \in Q$, $l_1, l_2, l_3 \in \{0, 1\}$ (note that this is the place where the point 2 of the definition of LTTM is used). Additionally, all the rules of the form $t_1 \to t_2$ where $t_2 \to t_1$ occurs in (1) or (2) above.
5. The rules that enable the existence of the loops defined by $\phi_{\text{loop}}(i)$:
$$[1]\ \bot_g^i(x) \to \bot_g^{i'}(x), \qquad [2]\ \bot_g^2(x) \to \bot_g^1(x)$$
where $i = 1, 2, 3$ and $i' = (i \mod 3) + 1$.

Note that we repeat some rules to make the classification of the rules more intuitive.

Now that we see the rewriting system, we can prove one direction of the equivalence between the rewriting and the LTTMs.

Definition 10. (terms that satisfy ϕ_M)
We can now define the terms $t_1, t_2, t_3, t_{31}, t_{32}, t_{21}, t_{22}$ which are used to satisfy the formula ϕ_M:

- $t_1 = \text{e2t}(\rho_1, \ldots, \rho_n)$,
- $t_2 = \text{e2t}(\rho_1, \ldots, \rho_{n-1})[\gamma \leftarrow \bot_g^1(t)]$ where $\text{e2t}(\rho_1, \ldots, \rho_{n-1})|_\gamma = \bot_g(t)$,
- $t_3 = t_1[\gamma \leftarrow \bot_g^1(t)]$ where $t_1|_\gamma = \bot_g(t)$,
- $t_{21} = t_2[\gamma \leftarrow \bot_g^2(t)]$ where γ is such that $t_2|_\gamma = \bot_g^1(t)$,
- $t_{22} = t_2[\gamma \leftarrow \bot_g^3(t)]$ where γ is such that $t_2|_\gamma = \bot_g^1(t)$,
- $t_{31} = t_3[\gamma \leftarrow \bot_g^2(t)]$ where γ is such that $t_3|_\gamma = \bot_g^1(t)$,
- $t_{32} = t_3[\gamma \leftarrow \bot_g^3(t)]$ where γ is such that $t_3|_\gamma = \bot_g^1(t)$.

Note one particular difference between t_1 and t_2. The topmost configuration encoded in t_1 is ρ_n while the topmost configuration encoded in t_2 is ρ_{n-1}. Therefore, the reduction $t_2 \to^p t_1$ must relate the encoding of the configuration ρ_{n-1} to the encoding of ρ_n. The way t_1 and t_2 are defined implies that this relationship generalises to ρ_{i-1} and ρ_i for each $i = 1, \ldots, n$. The relationship indeed holds after the encoding as the steps of the machine can be simulated by the term rewriting with help of the rules in the group (4.) of Def. 9. This is also the place where the strength of the parallel rewriting is used. As no path to a configuration encoding is a prefix of a path to another configuration encoding, we can fire all the rewrites pertinent to a move of the machine M at the same time.

Lemma 1. (from runs of machines to witnesses of the formula)
If $\rho_1, \ldots, \rho_n$ is a run of an LTTM M then terms $t_1, t_2, t_3, t_{21}, t_{22}, t_{31}, t_{32}$ from Def. 10 substituted for $x_1, x_2, x_3, x_{21}, x_{22}, x_{31}, x_{32}$ respectively witness that the formula ϕ_M holds.

Proof. The proof is by a routine case analysis.

The reduction $t_1 \to^p t_1$ is guaranteed not to hold as the only rules which enable the self-rewriting are: in the group (3.) and (1.)[3]. At the same time none of the patterns in the left-hand sides of the rules occur in t_1. This proves that ϕ_{form} holds.

For the reductions in ϕ_{start}, the reduction $t_1 \to^p t_3$ is guaranteed by the rule (1.)[1]. The reduction $t_3 \to^p t_3$ is guaranteed by the rule (1.)[2]. The reduction $t_3 \to^p t_1$ is impossible as the only way to dispose of the symbol $\bot_g^1$ is either to use the rule (1.)[3], but then the number of gs must increase, or to use a rule in (5.)[1], but then you cannot get back $\bot_g$. The positive rewrites in the subformula $\phi_{\text{loop}}(3)$ are possible because of the rules in (5.). The (negative) rewrite $t_3 \to^p t_{32}$ is impossible as there is no rule which is able to change $\bot_g^1$ to $\bot_g^3$. This proves that ϕ_{start} holds.

For the reductions in ϕ_{run}, the reduction $t_1 \to^p t_2$ is impossible as there is no way to change $g(\bot_g(s_1), s_2)$ in t_1 to $\bot_g^1(s_3)$ in t_2. Indeed, the only rewrite rule which has $\bot_g$ on the left-hand side and is not a self-rewrite rule is (1.)[1]. This, however, does not change the number of gs on the path to $\bot_g^1$. Therefore, this rule cannot change $g(\bot_g(s_1), s_2)$ to $\bot_g^1(s_3)$.

The reduction $t_2 \to^p t_1$ is possible because $t_2|_{(1^i)2}$ represents the configuration of M one step before the configuration represented by $t_1|_{(1^i)2}$ and the rules in the group (4.) can be applied at an appropriate position in t_2 to obtain the corresponding part of t_1 (see Fig. 2. Moreover, the rule (1.)[3] can be used to transform $\bot_g^1(s_4)$ to $g(\bot_g(s_4), s_4)$. The rewrites in the subformula $\phi_{\text{loop}}(2)$ hold for the reasons similar to the case of $\phi_{\text{loop}}(3)$. This proves that ϕ_{run} holds.

For the reductions in ϕ_{end}, the reduction $t_3 \to^p t_2$ is possible by the rules in (2.) while the reduction $t_2 \to^p t_3$ is impossible as the only rule which increases the number of occurrences of the symbol g is the rule (1.)[3], but this rule can be applied only at one place in t_2 (as there is only one occurrence of $\bot_g^1$), but then it is impossible to obtain t_3 as t_3 contains $\bot_g^1$ which is removed by the rule. Thus ϕ_{end} holds. The further details are left for the reader.

The lemma above establishes one direction of the relation between M and ϕ_M. In order to establish the other direction we have to define several forms of terms which can be enforced by certain graph structures expressed in the formula ϕ_M. We start with the forms which can be enforced by means of the subformula $x \not\to^p x$ (this constrains the term substituted for x so that certain local patterns are forbidden). The rules in Def. 9 cause that this leads to shapes close to encodings of configurations and runs. We start with a form which corresponds to configurations.

Definition 11. `(pseudo-configuration form)`
The set of terms in *pseudo-configuration* form is defined inductively as the smallest set that contains terms: $\bot_f$, and $f(c, t)$ where $c \in Q \times \{0, 1\} \cup \{0, 1\}$ and t is in pseudo-configuration form.

Now, we define the form which corresponds to runs of a machine.

Definition 12. (pseudo-run form)
The set of terms in *pseudo-run* form is defined inductively as the smallest set that contains terms:

1. $\perp_g(t)$ where t is in pseudo-configuration form,
2. $g(t_1, t_2)$ where the subterm t_1 is in pseudo-run form and t_2 is in pseudo-configuration form

and in which the pattern $g(g(x, f(\langle q, l\rangle, y)), f(\langle q, l\rangle, z))$ does not occur where $q \in Q$ and $l \in \{0, 1\}$.

The terms in *weak pseudo-run form* are defined as above, but the final requirement about the pattern is dropped. A term is in *(weak)* $\perp_g^1$*-pseudo-run* form when after replacing symbols $\perp_g^1$ with $\perp_g$ it is in (weak) pseudo-run form.

We have to consider the weak forms as the rules in the group (4.) in Def. 9 may transform a term in pseudo-run form into one where the forbidden pattern occurs. The forms with $\perp_g^1$ are introduced as the rule (1.)[1], which checks that the initial configuration occurs in a term, replaces the symbol $\perp_g$ with $\perp_g^1$.

The defined below x_1-form is the form which is exactly enforced by the subformula $x \not\rightarrow^p x$. The x_*-form is a general term to capture all the possible forms of terms that can be substituted for x_1, x_2, and x_3 in ϕ_M.

Definition 13. (x_1-form and x_*-form)
We say that a term t is in x_1*-form* when it is either a constant, or is in pseudo-configuration form, or is in pseudo-run form.

A term t is in x_**-form* when it is either in x_1-form or is in weak $\perp_g^1$-pseudo-run form.

We have now a bunch of technical lemmas that relate the topological structures in the formula ϕ_M with the forms defined above. The proofs of these lemmas are by a routine case analysis combined with induction. We reproduce one of the proofs here to give the reader the general idea.

We start with a characterisation of terms that cannot be reduced to themselves. This is expressed by the subformula ϕ_{form} of ϕ_M.

Lemma 2. (formula and pseudo-run form)
A term t satisfies the formula $x \not\rightarrow^p x$ iff t is in x_1-form.

The following lemma characterises both the reduction in formulae $\phi_{\text{loop}}(i)$ for $i = 2, 3$ and the reductions between x_1, x_3 and x_2. This lemma requires in its proof the use of the $\sim_M$ relation instead of $\rightarrow_M^*$.

Lemma 3. (loop and the pseudo-configuration form)
If t is in pseudo-configuration form then there are no t_1, t_2 such that the substitution $S = \{x_* := t, x_{*1} := t_1, x_{*2} := t_2\}$ satisfies the formula

$$x_* \rightarrow^p x_{*1} \wedge x_{*1} \rightarrow^p x_{*2} \wedge x_{*2} \rightarrow^p x_* \wedge x_* \not\rightarrow^p x_{*2}.$$

The following lemma characterises the reductions of pseudo-configuration and weak pseudo-run terms.

Lemma 4. (reduction of the pseudo-configuration and -run forms)
(1) Let t_1 be in pseudo-configuration form. If $t_1 \to^p t_2$ then t_2 is in pseudo-configuration form.
(2) Let t_1 be in weak pseudo-run form. If $t_1 \to^p t_2$ then t_2 is in weak pseudo-run form or in weak $\bot_g^1$-pseudo-run form.

The following lemma gives us a tool to enforce that the reduction $x_1 \to^p x_3$ results in a term substituted for x_3 which contains $\bot_g^1$.

Lemma 5. (reduction in a loop)
If t is in weak pseudo-run form then there are no t_1, t_2 such that the substitution $S = \{x_* := t, x_{*1} := t_1, x_{*2} := t_2\}$ satisfies the formula $\phi_{\text{loop}}(*)$.

The following lemma gives us a tool to enforce that the reduction $x_2 \to^p x_3$ does not lose $\bot_g^1$.

Lemma 6. (on x_*-form)
Let $t_1 \to^p t_2$. If t_1 is in x_*-form and there are t_{21} and t_{22} such that the substitution $S = \{x_* := t_2, x_{*1} := t_{21}, x_{*2} := t_{22}\}$ satisfies the formula $\phi_{\text{loop}}(*)$ then t_2 is in weak $\bot_g^1$-pseudo-run form.

The following lemma constructs an eq-run of M based on the witnesses for ϕ_M.

Lemma 7. (from witnesses of the formula to eq-runs of machines)
Let R be a rewriting system from Def. 9 constructed for an LTTM M. If the formula ϕ_M from Def. 5 holds for the rewriting with R, then there is an eq-run of M.

Proof. Let $t_1, t_2, t_3, t_{21}, t_{22}, t_{31}, t_{32}$ be the witnesses that the formula ϕ_M holds. The terms t_1, t_2 and t_3 are pairwise different as making any of the pair equal would result in impossible situation that both $t \to^p t$ and $t \not\to^p t$ hold. Since $t_1 \not\to^p t_1$, we obtain by Lemma 2 that t_1 is in pseudo-run form or pseudo-configuration form, or is a constant. It cannot be a constant since we have $t_1 \to^p t_3$ and there is no rewriting rule which applied to a constant on the left-hand side. Moreover, t_1 cannot be in pseudo-configuration form as then Lemma 3 applied to t_1 would be contradicted by the rewrites between t_2 and t_3 enforced by ϕ_M. Thus, t_1 may only be in pseudo-run form.

Lemma 6 implies that t_2 and t_3 are in weak $\bot_g^1$-pseudo-run form. This implies that the reduction $t_1 \to^p t_3$ must be done with the use of the rule (1.)[1] from Def. 9. As t_2 is in the weak $\bot_g^1$-pseudo-run form and t_1 in pseudo-run form the reduction $t_2 \to^p t_1$ must use the rule (1.)[3] (the only rules which dispose of $\bot_g^1$ are (1.)[3] and 5[1], but the latter does not lead to the pseudo-run form). Since the rule (1.)[3] increases the number of gs on the leftmost branch, one of the rewrites $t_1 \to^p t_3 \to^p t_2 \to^p t_1$ must decrease it. The only rules that decrease the number of gs are the rules from (2.). This, however, can be used only in the rewriting $t_3 \to^p t_2$ as otherwise the use of the rule (1.)[3] in $t_2 \to^p t_1$ is impossible as well as the use of (1.)[1] in $t_1 \to^p t_3$.

Let γ be the address where a rule from (2.) is applied to t_3 in $t_3 \to^p t_2$. Let $t_i|_{\gamma \cdot 1^n 2} = u_i^n$ where $i = 1, 2, 3$. The terms u_i^n are in pseudo-configuration form

(we conclude that by Lemma 4(1) applied to u_1^n and then to u_3^n). This means that only the rules in the group (4.) can be applied to them. Additionally, we have the dependencies $u_1^i \to_*^p u_3^i = u_2^{i-1} \to_*^p u_1^{i-1}$ ($u_3^i = u_2^{i-1}$ as the rule from (2.) is applied at γ in $t_3 \to^p t_2$, and γ is a prefix of all the paths which point to u_j^i). This justifies that $u_1^i \to_*^p u_1^{i-1}$. Let $\gamma \cdot 1^{m+1}$ be the address of $\bot_g^1$ in t_3. Note that u_1^m is the encoding of an initial configuration, as the rule (1.)[1] is used in $t_1 \to^p t_3$. The term u_1^m is a properly formed encoding of a configuration which is guaranteed by the rule (1.)[1] applied in $t_1 \to^p t_3$. By induction on i we obtain that all pseudo-configuration forms u_1^{m-i} are proper encodings as well as that $u_1^{m-i-1} \sim_M u_1^{m-i}$. This is because the rules in (4.) directly encode forward and backward transitions of M. Further, we obtain by the downwards induction on i that each u_1^i is in the relation $\sim_M$ with u_1^m. The application of a reduction from (2.) in $t_3 \to^p t_2$ enforces that u_3^0 represents a final configuration of M by the point (3) of Def. 3. In this way, we obtain from a solution of the formula ϕ_M two configurations ρ_1, ρ_2 such that ρ_1 is the initial one, ρ_2 is the final one and $\rho_1 \sim_M \rho_2$ holds.

As the result of this lemma and Lemma 1 we obtain the theorem:

Theorem 2. `(translation between machines and rewrite systems)` *For each LTTM M and its initial configuration ρ there is a (left-linear) rewrite system R such that $\rho \sim_M \rho'$ for some final configuration ρ' iff the formula ϕ_M holds.*

Theorem 3. `(the undecidability of one-step parallel rewriting theory)`
There is no algorithm that given a signature Σ, an existential formula ϕ, and a rewrite system R over Σ can decide if ϕ is satisfied in the parallel rewriting structure $\mathcal{A}_R$ from Def. 1. In particular, the existential fragment of the theory of one-step parallel rewriting is undecidable.

This holds even if ϕ is restricted to contain at least 7 quantifiers and R to be left-linear.

4 Discussion

The rewrite rules of the form $t \to t$ in [Tre98] have been criticised. The current paper uses such rules. It is an open question if this can be avoided in case of the theory we consider here. Still, I conjecture this can be avoided with help of more complicated graphs specified by the formula ϕ_M. I also conjecture that a more complicated graphs would be able to prove that the theory is undecidable when positive atomic formulae are used exclusively.

Vorobyov in [Vor02] distinguishes weak and strong undecidability for this kind of theory. The strong one holds when the undecidability is proved for a fixed rewriting system while weak holds in other cases. In the sense, this paper presents the weak undecidability. However, the current construction works with a fixed formula. I conjecture that the strong version in the Vorobyov's sense also

holds for the existential fragment of the theory. I did not prove this strong version as the plans of such constructions that I could develop were very complicated.

The paper [CSTT99] uses a one-step rewriting theory with $\rightarrow$ predicate replaced by predicates $\rightarrow^r$, one for each rewriting rule r. The main result of the current paper can be straightforwardly adapted to this version of the theory even in case of Vorobyov's strong sense.

As the parallel term rewriting does not differ from the traditional one on unary signatures, we also obtain the decidability in the case considered in [Jac96]. We conjecture that the cases of the theory considered in [NPR97, CSTT99] are decidable, too.

The notion of concurrent rewriting defined in [GKM87] is more liberal and allows rewrites in case one occurrence of a redex is under another one. The current proof can be adapted to that notion, however the argument is more involved. The construction in the current paper does not adapt easily to the notions of maximal concurrent rewriting and numerically maximal concurrent rewriting defined in [GKM87]. However, we conjecture that with a bigger effort it can be accommodated to prove undecidability for these notions, too.

One more interesting point concerns the connection between the second-order unification and the rewriting. These problems are strongly related as shown in papers [NPR97, NTT00, LV00]. The positive existential theory of parallel one-step rewriting can be reduced to finding non-trivial solutions to special form flex-flex second-order unification equations. This can be achieved by a straightforward adaptation of the construction from [NPR97]. The current paper uses negative formulae so it does not prove the undecidability of the problem. This, however, rises the question if the positive fragment of the parallel one-step rewriting is decidable.

Acknowledgement

The author thanks Erik Poll, Pawel Urzyczyn and the anonymous referees for the help in preparation of the paper.

References

[AK96] Alouini, I., Kirchner, C.: Toward the concurrent implementation of computational systems. In: Hanus, M., Rodríguez-Artalejo, M. (eds.) ALP 1996. LNCS, vol. 1139, pp. 1–31. Springer, Heidelberg (1996)

[Ass00] Assmann, U.: Graph rewrite systems for program optimization. ACM Trans. Program. Lang. Syst. 22(4), 583–637 (2000)

[CSTT99] Caron, A.-C., Seynhaeve, F., Tison, S., Tommasi, M.: Deciding the satisfiability of quantifier free formulae on one-step rewriting. In: Narendran, P., Rusinowitch, M. (eds.) RTA 1999. LNCS, vol. 1631, pp. 103–117. Springer, Heidelberg (1999)

[GKM87] Goguen, J., Kirchner, C., Meseguer, J.: Concurrent term rewriting as a model of computation. In: Fasel, J.H., Keller, R.M. (eds.) Graph Reduction 1986. LNCS, vol. 279, pp. 53–93. Springer, Heidelberg (1987)

[Jac96] Jacquemard, F.: Automates d'arbres et Réécriture de termes. PhD thesis, Université de Paris-Sud (1996)

[LR99] Limet, S., Réty, P.: A new result about the decidability of the existential one-step rewriting theory. In: Narendran, P., Rusinowitch, M. (eds.) RTA 1999. LNCS, vol. 1631, pp. 118–132. Springer, Heidelberg (1999)

[LV00] Levy, J., Veanes, M.: On the undecidability of second-order unification. Information and Computation 159(1-2), 125–150 (2000)

[Mar97] Marcinkowski, J.: Undecidability of the first order theory of one-step right ground rewriting. In: Comon, H. (ed.) RTA 1997. LNCS, vol. 1232. Springer, Heidelberg (1997)

[MOM02] Martí-Oliet, N., Meseguer, J.: Rewriting logic: roadmap and bibliography. Theor. Comput. Sci. 285(2), 121–154 (2002)

[NPR97] Niehren, J., Pinkal, M., Ruhrberg, P.: On equality up-to constraints over finite trees, context unification and one-step rewriting. In: McCune, W. (ed.) CADE 1997. LNCS (LNAI), vol. 1249, pp. 34–48. Springer, Heidelberg (1997)

[NTT00] Niehren, J., Treinen, R., Tison, S.: On rewrite constraints and context unification. Information Processing Letters 74(1-2), 35–40 (2000)

[STT97] Seynhaeve, F., Tommasi, M., Treinen, R.: Grid structures and undecidable constraint theories. In: Bidoit, M., Dauchet, M. (eds.) CAAP 1997, FASE 1997, and TAPSOFT 1997. LNCS, vol. 1214, pp. 357–368. Springer, Heidelberg (1997)

[STTT01] Seynhaeve, F., Tison, S., Tommasi, M., Treinen, R.: Grid structures and undecidable constraint theories. Theoretical Computer Science 1-2(258), 453–490 (2001)

[Tis90] Tison, S.: Automates comme outil de décision dans les arbres. Dossier d'habilitation à diriger des recherches (December 1990) (in French)

[Tre96] Treinen, R.: The first-order theory of one-step rewriting is undecidable. In: Ganzinger, H. (ed.) RTA 1996. LNCS, vol. 1103, pp. 276–286. Springer, Heidelberg (1996)

[Tre98] Treinen, R.: The first-order theory of linear one-step rewriting is undecidable. Theoretical Computer Science 1-2(208), 149–177 (1998)

[Vor97] Vorobyov, S.: The first-order theory of one step rewriting in linear noetheran systems is undecidable. In: Comon, H. (ed.) RTA 1997. LNCS, vol. 1232, pp. 254–268. Springer, Heidelberg (1997)

[Vor02] Vorobyov, S.: The undecidability of the first-order theories of one step rewriting in linear canonical systems. Inf. Comput. 174, 1–32 (2002)

Proving Confluence of Term Rewriting Systems Automatically

Takahito Aoto, Junichi Yoshida, and Yoshihito Toyama

RIEC, Tohoku University
{aoto,yoshida,toyama}@nue.riec.tohoku.ac.jp

Abstract. We have developed an automated confluence prover for term rewriting systems (TRSs). This paper presents theoretical and technical ingredients that have been used in our prover. A distinctive feature of our prover is incorporation of several divide–and–conquer criteria such as those for commutative (Toyama, 1988), layer-preserving (Ohlebusch, 1994) and persistent (Aoto & Toyama, 1997) combinations. For a TRS to which direct confluence criteria do not apply, the prover decomposes it into components and tries to apply direct confluence criteria to each component. Then the prover combines these results to infer the (non-)confluence of the whole system. To the best of our knowledge, an automated confluence prover based on such an approach has been unknown.

1 Introduction

Termination and confluence are considered to be central properties of term rewriting systems. Recently, automation of termination proving has been widely investigated in the literature. On the other hand, automation of confluence proving has not been well-known. Numerous results have been obtained on proving the confluence of term rewriting systems [5,7,11,12,14,16,17,18,20], but little work is reported on automation or an integration of these results on a confluence prover. This motivates us to develop a fully-automated confluence prover. A distinctive feature of our prover is incorporation of several divide–and–conquer criteria such as those for commutative [16], layer-preserving [9] and persistent [2] combinations. We present theoretical and technical ingredients that have been used in our prover, design of the system, and some experimental results.

2 Preliminaries

This section fixes some notions and notations used in this paper. We refer to [3] for omitted definitions.

The sets of *function symbols* and *variables* are denoted by $\mathcal{F}$ and V. We denote by $\mathcal{F}(t)$ and $\mathrm{V}(t)$ the sets of function symbols and variables occurring in a term t. The symbol in t at a position p is written as $t(p)$. The root position is denoted by ϵ. The (proper) subterm relation is denoted by $\trianglelefteq$ ($\triangleleft$). A *rewrite rule* $l \to r$ is a pair of terms l and r such that $l \notin V$ and $\mathrm{V}(l) \supseteq \mathrm{V}(r)$. It is

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 93–102, 2009.
© Springer-Verlag Berlin Heidelberg 2009

collapsing if $r \in V$ and *left-linear* if no variable occurs more than twice in l. Rewrite rules are identified modulo variable renaming. A *term rewriting system* (*TRS* for short) is a finite set of rewrite rules. We put $\mathcal{F}(l \to r) = \mathcal{F}(l) \cup \mathcal{F}(r)$, $\mathrm{V}(l \to r) = \mathrm{V}(l) \cup \mathrm{V}(r)$ and $\mathcal{F}(\mathcal{R}) = \bigcup_{l \to r \in \mathcal{R}} \mathcal{F}(l \to r)$.

A rewrite step $s \to_{\mathcal{R},p} t$ is defined by $s/p = l\sigma$ and $t = s[r\sigma]_p$ for some $l \to r \in \mathcal{R}$ and substitution σ. We omit subscripts $_{\mathcal{R}}$ and/or $_p$ if they are not important. The subterm s/p of s is the *redex* of the rewrite step $s \to_p t$. A term s is in *normal form* if $s \to t$ for no term t. The set of normal forms is denoted by $\mathrm{NF}(\mathcal{R})$. A rewrite step $s \to_p t$ is said to be *innermost* when any $u \lhd s/p$ is in normal form. An innermost rewrite step is written as $s \to^i t$. The reflexive transitive closure (reflexive closure) of a relation $\to$ is denoted by $\xrightarrow{*}$ (resp. $\xrightarrow{=}$). We define $\xrightarrow{*}{}^i$ and $\xrightarrow{=}{}^i$ similarly. A term s is in *head normal form* if there exists no redex t such that $s \xrightarrow{*} t$. The set of head normal forms is denoted by $\mathrm{HNF}(\mathcal{R})$. A *normal form* (*head normal form*) of s is a term $t \in \mathrm{NF}(\mathcal{R})$ (resp. $t \in \mathrm{HNF}(\mathcal{R})$) such that $s \xrightarrow{*} t$. A normal form of s is denoted by $s{\downarrow}$. Similarly, an *innermost normal form* of s is denoted by $s{\downarrow^i}$ which is obtained by replacing $\to$ by $\to^i$. We use $\circ$ for the composition of relations. Terms s and t are *joinable* (*innermost joinable*) if $s \xrightarrow{*} \circ \xleftarrow{*} t$ ($s \xrightarrow{*}{}^i \circ \xleftarrow{*}{}^i t$, respectively.) A TRS $\mathcal{R}$ is *confluent* (*locally confluent*) or *Church–Rosser* if $s \xleftarrow{*} \circ \xrightarrow{*} t$ (resp. $s \leftarrow \circ \to t$) implies s and t are joinable. A TRS $\mathcal{R}$ is *terminating* (*innermost-terminating*) if there exists no infinite rewrite sequence $s_1 \to s_2 \to s_3 \to \cdots$ ($s_1 \to^i s_2 \to^i s_3 \to^i \cdots$, respectively). A *development rewrite step* $\multimap$ [9] is inductively defined as follows: (1) $\xrightarrow{=} \subseteq \multimap$, (2) $s_i \multimap t_i$ $(1 \le i \le n)$ implies $f(s_1, \ldots, s_n) \multimap f(t_1, \ldots, t_n)$, or (3) $s \multimap t$ if $s/p = l\sigma$, $s[r\sigma']_p = t$ and $\sigma(x) \multimap \sigma'(x)$ for any $x \in V$.

Let s, t be terms whose variables are disjoint. The term s *overlaps* on t (at position p) when there exists a non-variable subterm $u = t/p$ of t such that u and s are unifiable. Let $l_1 \to r_1$ and $l_2 \to r_2$ be rewrite rules w.l.o.g. whose variables are disjoint. Suppose that l_1 overlaps on l_2 at position p. Let σ be the most general unifier of l_1 and l_2/p. Then the term $l_2[l_1]\sigma$ yields a peak $l_2[r_1]\sigma \leftarrow l_2[l_1]\sigma \to r_2\sigma$. The pair $\langle l_2[r_1]\sigma, r_2\sigma \rangle$ is called the *critical pair* obtained by the overlap of $l_1 \to r_1$ on $l_2 \to r_2$ at position p. In the case of self-overlap (i.e. when $l_1 \to r_1$ and $l_2 \to r_2$ are identical modulo renaming), we do not consider the case $p = \epsilon$. It is an *outer critical pair* if $p = \epsilon$; else an *inner critical pair*. The set $\mathrm{CP}_{in}(l_1 \to r_1, l_2 \to r_2)$ ($\mathrm{CP}_{out}(l_1 \to r_1, l_2 \to r_2)$) is the set of inner critical pairs (outer critical pairs, respectively) obtained by the overlap of $l_1 \to r_1$ on $l_2 \to r_2$. For TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$ the set $\mathrm{CP}_{in}(\mathcal{R}_1, \mathcal{R}_2)$ of inner critical pairs are defined by $\bigcup_{l_1 \to r_1 \in \mathcal{R}_1} \bigcup_{l_2 \to r_2 \in \mathcal{R}_2} \mathrm{CP}_{in}(l_1 \to r_1, l_2 \to r_2)$. $\mathrm{CP}_{out}(\mathcal{R}_1, \mathcal{R}_2)$ is defined similarly. We set $\mathrm{CP}(\mathcal{R}_1, \mathcal{R}_2) = \mathrm{CP}_{in}(\mathcal{R}_1, \mathcal{R}_2) \cup \mathrm{CP}_{out}(\mathcal{R}_1, \mathcal{R}_2)$, $\mathrm{CP}_{\alpha}(\mathcal{R}) = \mathrm{CP}_{\alpha}(\mathcal{R}, \mathcal{R})$ ($\alpha \in \{in, out\}$) and $\mathrm{CP}(\mathcal{R}) = \mathrm{CP}_{in}(\mathcal{R}) \cup \mathrm{CP}_{out}(\mathcal{R})$. A TRS $\mathcal{R}$ is *overlay* if $\mathrm{CP}_{in}(\mathcal{R}) = \emptyset$. Note that $\langle s, t \rangle \in \mathrm{CP}_{out}(\mathcal{R})$ iff $\langle t, s \rangle \in \mathrm{CP}_{out}(\mathcal{R})$.

3 Direct Methods

In this section, we explain direct methods employed in our confluence prover.

Proposition 1 (Knuth–Bendix's criterion[8]). A terminating TRS $\mathcal{R}$ is confluent if and only if $s\downarrow = t\downarrow$ for all $\langle s, t\rangle \in \mathrm{CP}(\mathcal{R})$.

Thus it is decidable whether a terminating TRS $\mathcal{R}$ is confluent or not. Hence termination proving take an important part of the confluence proving procedure. Termination, however, is undecidable property of TRSs and we should take into account the case where the prover fails to prove termination of terminating TRSs.

Since termination implies innermost-termination, it is natural to expect that innermost-termination proving is more powerful than termination proving. This is especially true for recent termination tools based on dependency pairs (see e.g. [4]). This motivates the following criterion mentioned by Ohlebusch [10] pp.125–126, which can be easily proved by Theorem 3.23 of Gramlich [6].

Theorem 1 (Gramlich–Ohlebusch's criterion). For innermost-terminating overlay TRS $\mathcal{R}$, $\mathcal{R}$ is confluent if and only if $s\downarrow^i = t\downarrow^i$ for all $\langle s, t\rangle \in \mathrm{CP}(\mathcal{R})$.

Thus for overlay TRSs, one can safely switch the termination proof to the innermost-termination proof and try Gramlich–Ohlebusch's criterion instead of Knuth–Bendix's criterion.

When our confluence prover fails to detect (innermost) termination, our prover next checks several sufficient confluence conditions.

Proposition 2 (Huet–Toyama–van Oostrom's criterion[20]). A left-linear TRS $\mathcal{R}$ is confluent if (1) $s \multimap t$ for all $\langle s, t\rangle \in \mathrm{CP}_{in}(\mathcal{R})$ and (2) $s \multimap \circ \overset{*}{\leftarrow} t$ for all $\langle s, t\rangle \in \mathrm{CP}_{out}(\mathcal{R})$.

Because $u \overset{*}{\rightarrow} v$ is undecidable in general, we use $u \multimap v$ instead, i.e. our prover checks (2′) $s \multimap \circ \leftarrow\!\circ\!\!-\!\!\circ\!\!- t$ for all $\langle s, t\rangle \in \mathrm{CP}_{out}(\mathcal{R})$ in the place of (2). The check of the following criterion is approximated in a similar way.

Proposition 3 (Huet's strong-closedness criterion[7]). A linear TRS $\mathcal{R}$ is confluent if $s \overset{=}{\rightarrow} \circ \overset{*}{\leftarrow} t$ and $s \overset{*}{\rightarrow} \circ \overset{=}{\leftarrow} t$ for all $\langle s, t\rangle \in \mathrm{CP}(\mathcal{R})$.

So far there is no mechanism of detecting non-confluence of non-terminating TRSs. Therefore, we add a simple non-confluence check based on the following easy observation.

Theorem 2 (A simple non-confluence criterion). If there exist terms s', t' such that $s \overset{*}{\rightarrow} s'$ and $t \overset{*}{\rightarrow} t'$ for some $\langle s, t\rangle \in \mathrm{CP}(\mathcal{R})$ which satisfy either (1) $s', t' \in \mathrm{NF}(\mathcal{R})$ and $s' \neq t'$; or (2) $s', t' \in \mathrm{HNF}(\mathcal{R})$ and $s'(\epsilon) \neq t'(\epsilon)$. Then $\mathcal{R}$ is not confluent.

Since it is undecidable whether a term s is in head normal form, our prover checks a simple sufficient criterion $s(\epsilon) \notin \{l(\epsilon) \mid l \rightarrow r \in \mathcal{R}\}$ instead.

4 Divide and Conquer Methods

When all of direct methods fail to prove the (non-)confluence of the TRS, our prover next tries to infer it using divide–and–conquer approach. Several criteria to infer the (non-)confluence of a TRS from that of its subsystems are known [1,2,9,15,16,17].

4.1 Persistent Decomposition

The first decomposition method our prover tries is the one based on the persistency [23], which is an extension of the direct-sum decomposition [15].

Definition 1 (persistent property[23]). A property $\mathcal{P}$ is said to be *persistent* if $\mathcal{P}$ holds for a many-sorted TRS $\mathcal{R}$ if and only if $\mathcal{P}$ holds for its underlying unsorted TRS $\Theta(\mathcal{R})$. Here Θ is the operation that forget the sort information.

Proposition 4 (persistency of confluence[2]). Confluence is a persistent property of TRSs.

Let us describe how persistency of confluence is used to show the confluence of a TRS from its subsystems.

Let $\mathcal{S}$ be a set of sorts. A sort attachment τ is a mapping $\mathcal{F} \to \mathcal{S}^*$ such that $\mathrm{arity}(f) = n$ implies $\tau(f) \in \mathcal{S}^{n+1}$, which is written as $f : \alpha_1 \times \cdots \times \alpha_n \to \alpha_0$. A sort attachment τ is consistent with a TRS $\mathcal{R}$ if for any $l \to r \in \mathcal{R}$, l and r are well-sorted under τ with the same sort. For any term t well-sorted under τ, let t^τ be the many-sorted term sorted by τ. Any sort attachment τ consistent with $\mathcal{R}$ induces a many-sorted TRS $\mathcal{R}^\tau = \{l^\tau \to r^\tau \mid l \to r \in \mathcal{R}\}$. Let us denote by T^τ the set of many-sorted terms, by T^τ_α the set of many-sorted terms of sort α, and let $\mathrm{T}^\tau_{\trianglelefteq\alpha} = \{t \in \mathrm{T}^\tau \mid t \trianglelefteq u \text{ for some } u \in \mathrm{T}^\tau_\alpha\}$. For any set A of terms closed under rewrite steps, let us write $\mathrm{Conf}(A)$ iff $s \to^* \circ \leftarrow^* t$ for any $s, t \in A$ such that $s \leftarrow^* \circ \to^* t$. By persistency, $\mathrm{Conf}(\mathrm{T}(\mathcal{F}, V))$ iff $\mathrm{Conf}(\mathrm{T}^\tau)$ iff $\mathrm{Conf}(\mathrm{T}^\tau_\alpha)$ for any sort α. Because any rewrite rule that can be applied to a term of sort α is in $\mathcal{R}^\tau \cap (\mathrm{T}^\tau_{\trianglelefteq\alpha})^2$, we have $\mathrm{Conf}(\mathrm{T}^\tau_\alpha)$ iff $\mathcal{R}^\tau \cap (\mathrm{T}^\tau_{\trianglelefteq\alpha})^2$ is confluent. Since the confluence of any set A of unsorted terms implies confluence of $\{t^\tau \mid t \in A\}$, $\mathcal{R}$ is confluent iff $\Theta(\mathcal{R}^\tau \cap (\mathrm{T}^\tau_{\trianglelefteq\alpha})^2)$ is confluent for any $\alpha \in \mathcal{S}$. Thus the following persistent criterion is obtained.

Definition 2 (persistent decomposition). Let $\mathcal{R}$ be a TRS and τ a sort attachment consistent with $\mathcal{R}$. Then $\max(\{\Theta(\mathcal{R}^\tau \cap (\mathrm{T}_{\trianglelefteq\alpha})^2) \mid \alpha \in \mathcal{S}\})$ is said to be a *persistent decomposition* of $\mathcal{R}$. Here max is the operation of taking maximal (w.r.t. subset relation) sets. We write $\mathcal{R} = \mathcal{R}_1 \oplus^\tau \cdots \oplus^\tau \mathcal{R}_n$ if $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ is a persistent decomposition of $\mathcal{R}$. A persistent decomposition is said to be *minimal* if each components has no persistent decomposition.

Theorem 3 (persistent criterion). A TRS $\mathcal{R} = \mathcal{R}_1 \oplus^\tau \cdots \oplus^\tau \mathcal{R}_n$ is confluent if and only if so is each $\mathcal{R}_i$.

Example 1 (confluence proof by persistency decomposition). Let

$$\mathcal{R}_a = \left\{ \begin{array}{lll} \mathsf{f}(\mathsf{a}(x), x) \to \mathsf{f}(x, \mathsf{a}(x)) & & \mathsf{g}(\mathsf{b}(x), y) \to \mathsf{g}(\mathsf{a}(\mathsf{a}(x)), y) \\ \mathsf{f}(\mathsf{b}(x), x) \to \mathsf{f}(x, \mathsf{b}(x)) & & \mathsf{g}(\mathsf{c}(x), y) \to y \\ \mathsf{a}(x) \to \mathsf{b}(x) & & \end{array} \right\}.$$

The direct methods do not apply to $\mathcal{R}_a$, because $\mathcal{R}_a$ is neither terminating nor left-linear. Consider the attachment τ on $\mathcal{S} = \{0, 1, 2\}$ such that $\mathsf{f} : 0 \times 0 \to 1$,

$\mathsf{a} : 0 \to 0$, $\mathsf{b} : 0 \to 0$, $\mathsf{c} : 0 \to 0$, and $\mathsf{g} : 0 \times 2 \to 2$. Then we have the following persistent decomposition of $\mathcal{R}_a$:

$$\mathcal{R}_{a1} = \left\{ \begin{array}{ll} \mathsf{f}(\mathsf{a}(x), x) & \to \mathsf{f}(x, \mathsf{a}(x)) \\ \mathsf{f}(\mathsf{b}(x), x) & \to \mathsf{f}(x, \mathsf{b}(x)) \\ \mathsf{a}(x) & \to \mathsf{b}(x) \end{array} \right\}, \quad \mathcal{R}_{a2} = \left\{ \begin{array}{lll} \mathsf{a}(x) & \to & \mathsf{b}(x) \\ \mathsf{g}(\mathsf{b}(x), y) & \to & \mathsf{g}(\mathsf{a}(\mathsf{a}(x)), y) \\ \mathsf{g}(\mathsf{c}(x), y) & \to y & \end{array} \right\}.$$

$\mathcal{R}_{a1}$ is confluent by Knuth–Bendix's criterion and $\mathcal{R}_{a2}$ is confluent by Huet–Toyama–van Oostrom's criterion. Thus the confluence of $\mathcal{R}_a$ follows from the persistency of confluence. □

Since a most general sort attachment consistent with a TRS is unique (modulo renaming of sorts), we know that a minimal persistent decomposition of $\mathcal{R}$ is unique. Since the smaller a component of persistent decomposition becomes, the easier it becomes to prove confluence of that component. Thus it suffices to compute the minimal persistent decomposition and try to prove (non-)confluence of each components. Furthermore, since the persistency is a necessary and sufficient criterion, if non-confluence is detected in any component then non-confluence of the whole TRS is concluded.

4.2 Layer-Preserving Decomposition

The second decomposition method our prover tries is the one based on the (composable) layer-preserving combination [9]. In what follows, $\mathcal{D}(l \to r) = \{l(\epsilon), r(\epsilon)\}$, $\setminus$ and $\uplus$ stand for the set difference and disjoint union.

Definition 3 (layer-preserving[9]). A pair $\langle \mathcal{R}_1, \mathcal{R}_2 \rangle$ of TRSs is said to be a *layer-preserving* combination if there exists a partition $\mathcal{D}_1 \uplus \mathcal{D}_2 \uplus \mathcal{C}$ of function symbols such that (1) for any $\alpha \in \mathcal{R}_1 \setminus \mathcal{R}_2$, $\mathcal{D}(\alpha) \subseteq \mathcal{D}_1$ and $\mathcal{F}(\alpha) \subseteq \mathcal{D}_1 \cup \mathcal{C}$ (2) for any $\alpha \in \mathcal{R}_2 \setminus \mathcal{R}_1$, $\mathcal{D}(\alpha) \subseteq \mathcal{D}_2$ and $\mathcal{F}(\alpha) \subseteq \mathcal{D}_2 \cup \mathcal{C}$ (3) for any $\alpha \in \mathcal{R}_1 \cap \mathcal{R}_2$, $\mathcal{F}(\alpha) \subseteq \mathcal{C}$. If $\langle \mathcal{R}_1, \mathcal{R}_2 \rangle$ is a layer-preserving combination the union $\mathcal{R}_1 \cup \mathcal{R}_2$ is denoted by $\mathcal{R}_1 \Cup \mathcal{R}_2$.

Based on the definition of layer-preserving combination given above we define the layer-preserving decomposition as follows.

Definition 4 (layer-preserving decomposition). A set $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ ($n \geq 1$, $\mathcal{R}_i \neq \emptyset$) of TRSs is said to be a *layer-preserving decomposition* of $\mathcal{R}$ (denoted by $\mathcal{R} = \mathcal{R}_1 \Cup \cdots \Cup \mathcal{R}_n$) if $\mathcal{R} = \mathcal{R}_1 \cup \cdots \cup \mathcal{R}_n$ and $\langle \mathcal{R}_i, \mathcal{R}_j \rangle$ is a layer-preserving combination for $i \neq j$. A layer-preserving decomposition is said to be *minimal* if each component has no layer-preserving decomposition.

Proposition 5 (layer-preserving criterion[9]). A TRS $\mathcal{R} = \mathcal{R}_1 \Cup \cdots \Cup \mathcal{R}_n$ is confluent if and only if so is each $\mathcal{R}_i$.

Example 2 (confluence proof by layer-preserving decomposition). Let

$$\mathcal{R}_b = \left\{ \begin{array}{lll} \mathsf{f}(x, \mathsf{a}(\mathsf{g}(x))) & \to \mathsf{g}(\mathsf{f}(x, x)) & \mathsf{a}(x) \to x \\ \mathsf{f}(x, \mathsf{g}(x)) & \to \mathsf{g}(\mathsf{f}(x, x)) & \mathsf{h}(x) \to \mathsf{h}(\mathsf{a}(\mathsf{h}(x))) \end{array} \right\}$$

It is clear that the direct methods do not apply to $\mathcal{R}_b$. Taking the partition of function symbols as $\mathcal{D}_1 = \{\mathsf{f}, \mathsf{g}\}, \mathcal{D}_2 = \{\mathsf{h}\}, \mathcal{C} = \{\mathsf{a}\}$, we have the following layer-preserving decomposition of $\mathcal{R}_b$:

$$\mathcal{R}_{b1} = \left\{ \begin{array}{ll} \mathsf{f}(x, \mathsf{a}(\mathsf{g}(x))) & \to \mathsf{g}(\mathsf{f}(x, x)) \\ \mathsf{f}(x, \mathsf{g}(x)) & \to \mathsf{g}(\mathsf{f}(x, x)) \\ \mathsf{a}(x) & \to x \end{array} \right\}, \ \mathcal{R}_{b2} = \left\{ \begin{array}{l} \mathsf{a}(x) \to x \\ \mathsf{h}(x) \to \mathsf{h}(\mathsf{a}(\mathsf{h}(x))) \end{array} \right\}.$$

$\mathcal{R}_{b1}$ is confluent by Knuth–Bendix's criterion and $\mathcal{R}_{b2}$ is confluent by Huet–Toyama–van Oostrom's criterion. Thus the confluence of $\mathcal{R}_b$ follows from the layer-preserving criterion. □

Note that the layer-preserving decomposition does not apply to $\mathcal{R}_a$ in Example 1 because of a collapsing rule. Similarly, the persistent decomposition does not apply to $\mathcal{R}_b$ in Example 2 because the only possible sort attachment is on a single sort. Thus, the two decompositions are incomparable.

One can show that a minimal layer-preserving decomposition is unique. Same as the case of persistent decomposition, it suffices to compute the minimal layer-preserving decomposition and if non-confluence is detected in any component then non-confluence of the whole TRS is concluded.

4.3 Commutative Decomposition

The third and last decomposition method our prover tries is the one based on commutation [13].

Definition 5 (commutation[13]). TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$ *commute* if ${}^{*}\!\leftarrow_{\mathcal{R}_1} \circ {}^{*}\!\rightarrow_{\mathcal{R}_2} \subseteq {}^{*}\!\rightarrow_{\mathcal{R}_2} \circ {}^{*}\!\leftarrow_{\mathcal{R}_1}$. If $\mathcal{R}_1$ and $\mathcal{R}_2$ commute then their union $\mathcal{R}_1 \cup \mathcal{R}_2$ is denoted by $\mathcal{R}_1 \sqcup \mathcal{R}_2$.

Definition 6 (commutative decomposition). A set $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ ($n \geq 1$, $\mathcal{R}_i \neq \emptyset$) of TRSs is said to be a *commutative decomposition* of $\mathcal{R}$ (denoted by $\mathcal{R} = \mathcal{R}_1 \sqcup \cdots \sqcup \mathcal{R}_n$) if $\mathcal{R} = \mathcal{R}_1 \cup \cdots \cup \mathcal{R}_n$ and $\mathcal{R}_i, \mathcal{R}_j$ commute for $i \neq j$. A commutative decomposition is said to be *minimal* if each component has no commutative decomposition.

Proposition 6 (commutativity criterion[13]). A TRS $\mathcal{R} = \mathcal{R}_1 \sqcup \cdots \sqcup \mathcal{R}_n$ is confluent if so is each $\mathcal{R}_i$.

Since the commutativity criterion is merely a sufficient criterion, unlike previous two decompositions, it can not be used to infer the non-confluence from that of its subsystems. Moreover, since non-left-linear rules destroy commutativity, the commutative decomposition is restricted to left-linear TRSs.

Commutativity of TRSs is undecidable in general but a sufficient condition is known [16]. Our prover employs a slightly more general condition which is obtained by extending the proof of [20] along the line of [16][1].

[1] The detailed proof can be found in [22].

Proposition 7 (sufficient condition for commutativity[22]). Left-linear TRSs $\mathcal{R}_1$ and $\mathcal{R}_2$ commute if (1) $s \multimap_{\mathcal{R}_2} t$ for any $\langle s,t\rangle \in \mathrm{CP}_{in}(\mathcal{R}_1,\mathcal{R}_2)$, and (2) $s \multimap_{\mathcal{R}_1} \circ \overset{*}{\leftarrow}_{\mathcal{R}_2} t$ for any $\langle t,s\rangle \in \mathrm{CP}(\mathcal{R}_2,\mathcal{R}_1)$.

Condition (2) is undecidable and our prover uses a condition $s \multimap_{\mathcal{R}_1} \circ \hookleftarrow_{\mathcal{R}_2} t$ instead of the condition $s \multimap_{\mathcal{R}_1} \circ \overset{*}{\leftarrow}_{\mathcal{R}_2} t$.

Example 3 (confluence proof by commutative decomposition). Let

$$\mathcal{R}_c = \left\{ \begin{array}{lll} \mathsf{f}(x) & \to \mathsf{g}(x) & \mathsf{f}(x) \to \mathsf{h}(\mathsf{f}(x)) \\ \mathsf{h}(\mathsf{f}(x)) & \to \mathsf{h}(\mathsf{g}(x)) & \mathsf{g}(x) \to \mathsf{h}(\mathsf{g}(x)) \end{array} \right\}.$$

Neither the direct methods nor the previous decomposition methods apply to $\mathcal{R}_c$. One can divide $\mathcal{R}_c$ into the following $\mathcal{R}_{c1}$ and $\mathcal{R}_{c2}$ which commute from Proposition 7.

$$\mathcal{R}_{c1} = \left\{ \begin{array}{ll} \mathsf{f}(x) & \to \mathsf{g}(x) \\ \mathsf{h}(\mathsf{f}(x)) & \to \mathsf{h}(\mathsf{g}(x)) \end{array} \right\},\ \mathcal{R}_{c2} = \left\{ \begin{array}{ll} \mathsf{f}(x) & \to \mathsf{h}(\mathsf{f}(x)) \\ \mathsf{g}(x) & \to \mathsf{h}(\mathsf{g}(x)) \end{array} \right\}.$$

$\mathcal{R}_{c1}$ is confluent by Knuth–Bendix's criterion and $\mathcal{R}_{c2}$ is confluent by Huet–Toyama–van Oostrom's criterion. Thus, the confluence of $\mathcal{R}_c$ follows from the commutativity criterion. □

Contrast to the persistent or layer-preserving decompositions, a minimal commutative decomposition is not unique. Furthermore, unlike the previous two decompositions, it does not always hold that a smaller decomposition is more useful to prove confluence [22], i.e. there is an example that can be proved by a non-minimal commutative decomposition but minimal ones fail.

5 Implementation and Experiments

We have implemented a confluence prover ACP (Automated Confluence Prover) in SML/NJ; the length of codes is about 14,000 lines[2]. It has a command line interface which takes an argument to specify a filename containing a TRS specification in TPDB[3] format. Several options are supported so that partial combinations of decompositions can be tested. An external termination prover can be specified in the place of an internal termination prover.

The overview of the prover is illustrated in Figure 1. Procedure *Direct* consists of the direct methods explained in Section 1. If *Direct* fails (i.e. neither confluence nor non-confluence is detected), then the prover finds the minimal persistent decomposition $\mathcal{R} = \mathcal{R}_1 \oplus^{\tau} \cdots \oplus^{\tau} \mathcal{R}_n$. If the decomposition is not proper (i.e. $n = 1$) then the prover tries a minimal layer-preserving decomposition to $\mathcal{R} = \mathcal{R}_1$. If

[2] A heap image of ACP that can be loaded into an SML/NJ runtime system, examples used for the experiments, and details of experiments can be obtained from `http://www.nue.riec.tohoku.ac.jp/tools/acp/`

[3] `http://www.lri.fr/~marche/tpdb/`

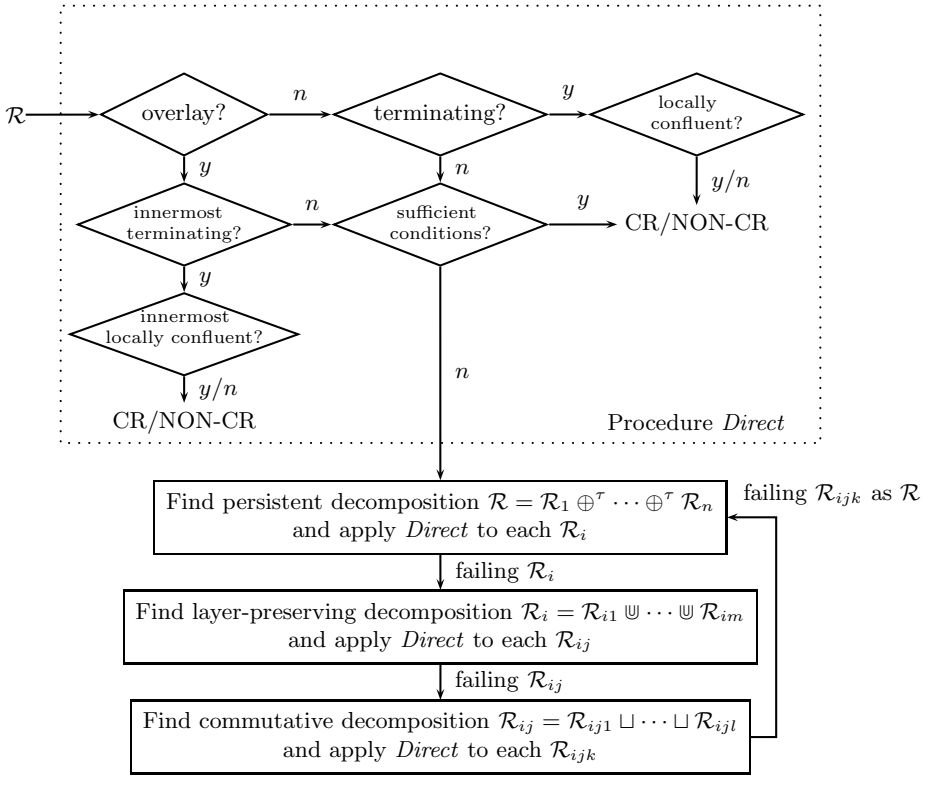


Fig. 1. Overview of ACP

the decomposition is proper (i.e. $n > 1$) then the prover applies the procedure *Direct* to each components $\mathcal{R}_1, \ldots, \mathcal{R}_n$. For each component $\mathcal{R}_i$ on which *Direct* fails, then the prover tries a minimal layer-preserving decomposition and so on. When the direct methods and successive three kinds of proper decompositions fail, the prover aborts the proof of (non-)confluence of (that component of) the system. Furthermore, if there is another possibility of decompositions then the prover backtracks and tries another decomposition.

For the experiment, we prepare a collection of 103 examples extracted from the literatures on confluence. We have tested confluence proving using various combinations of decomposition techniques. All tests have been performed in a PC equipped with Intel Xeon processors of 2.66GHz and a memory of 7GB.

The table below summarizes our experimental results. d, p, l, c, c' stand for direct methods, (minimal) persistent decomposition, (minimal) layer-preserving decomposition, minimal commutative decomposition, (possibly non-minimal) commutative decomposition, respectively.

	d	dp	dl	dc	dc'	dpl	dpc	dpc'	dlc	dlc'	dplc	dplc'
success (CR)	30	35	34	41	43	37	47	49	46	47	49	50
success (NON-CR)	13	13	13	13	13	13	13	13	13	13	13	13
failure	60	55	56	49	47	53	43	41	44	43	41	40
timeout (60 sec.)	0	0	0	0	0	0	0	0	0	0	0	0
total time (sec.)	4.1	4.4	4.8	7.0	33.6	4.9	8.1	30.9	7.6	34.2	8.1	34.4

It is seen that decomposition techniques are effective to prove confluence, although it is ineffective to prove non-confluence. Each decomposition techniques succeeds to prove confluence of some different examples. Commutative decomposition is costly but most powerful of three decompositions.

6 Conclusion

We have presented an automated confluence prover ACP for TRSs, in which divide–and–conquer approach based on the persistent, layer-preserving, commutative decompositions is employed. To the best of our knowledge, an automated confluence prover based on such an approach has been unknown. We believe that our approach is useful to integrate different (non-)confluence criteria to prove the (non-)confluence of large systems.

The previous version of our confluence prover has been described in [22]. The new version is different in the following points in particular: new direct criteria (Gramlich–Ohlebusch's criterion, Huet's strong-closedness criterion, a simple non-confluence criterion) are added; the direct-sum decomposition is replaced with the persistent decomposition; the layer-preserving decomposition is added; the algorithm for computing commutative decompositions is changed to improve the efficiency.

There are still many confluence criteria which are not included in our prover—for example, stronger sufficient criteria for left-linear TRSs (e.g. [11,12,18]), the decreasing diagram technique (e.g. [19,21]) and decision procedures for some subclass of TRSs (e.g. [5,14]). It is our future work to include these criteria and make the system more powerful.

Acknowledgments

Thanks are due to anonymous referees for detailed and helpful comments. This work was partially supported by grants from JSPS, Nos. 19500003 and 20500002.

References

1. Aoto, T., Toyama, Y.: On composable properties of term rewriting systems. In: Hanus, M., Heering, J., Meinke, K. (eds.) ALP 1997 and HOA 1997. LNCS, vol. 1298, pp. 114–128. Springer, Heidelberg (1997)

2. Aoto, T., Toyama, Y.: Persistency of confluence. Journal of Universal Computer Science 3(11), 1134–1147 (1997)
3. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
4. Giesl, J., Thiemann, R., Schneider-Kamp, P.: Mechanizing and improving dependency pairs. Journal of Automated Reasoning 37(3), 155–203 (2006)
5. Godoy, G., Tiwari, A.: Confluence of shallow right-linear rewrite systems. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 541–556. Springer, Heidelberg (2005)
6. Gramlich, B.: Abstract relations between restricted termination and confluence property of rewrite systems. Fundamenta Informaticae 24, 3–23 (1995)
7. Huet, G.: Confluent reductions: abstract properties and applications to term rewriting systems. Journal of the ACM 27(4), 797–821 (1980)
8. Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebras. In: Leech, J. (ed.) Computational Problems in Abstract Algebra, pp. 263–297. Pergamon Press (1970)
9. Ohlebusch, E.: Modular Properties of Composable Term Rewriting Systems. PhD thesis, Universität Bielefeld (1994)
10. Ohlebusch, E.: Advanced Topics in Term Rewriting Systems. Springer, Heidelberg (2002)
11. Okui, S.: Simultaneous critical pairs and Church-Rosser property. In: Nipkow, T. (ed.) RTA 1998. LNCS, vol. 1379, pp. 2–16. Springer, Heidelberg (1998)
12. Oyamaguchi, M., Ohta, Y.: A new parallel closed condition for Church-Rosser of left-linear TRS's. In: Comon, H. (ed.) RTA 1997. LNCS, vol. 1232, pp. 187–201. Springer, Heidelberg (1997)
13. Rosen, B.K.: Tree-manipulating systems and Church-Rosser theorems. Journal of the ACM 20, 160–187 (1973)
14. Tiwari, A.: Deciding confluence of certain term rewriting systems in polynomial time. In: Proc. of LICS 2002, pp. 447–458. IEEE Computer Society Press, Los Alamitos (2002)
15. Toyama, Y.: On the Church-Rosser property for the direct sum of term rewriting systems. Journal of the ACM 34(1), 128–143 (1987)
16. Toyama, Y.: Commutativity of term rewriting systems. In: Fuchi, K., Kott, L. (eds.) Programming of Future Generation Computers II, pp. 393–407. North-Holland, Amsterdam (1988)
17. Toyama, Y.: Confluent term rewriting systems (invited talk). In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, p. 1. Springer, Heidelberg (2005), `http://www.nue.riec.tohoku.ac.jp/user/toyama/slides/toyama-RTA05.pdf`
18. Toyama, Y., Oyamaguchi, M.: Conditional linearization of non-duplicating term rewriting systems. IEICE Trans. Information and Systems E84-D(5), 439–447 (2001)
19. van Oostrom, V.: Confluence by decreasing diagrams. Theoretical Computer Science 126(2), 259–280 (1994)
20. van Oostrom, V.: Developing developments. Theoretical Computer Science 175(1), 159–181 (1997)
21. van Oostrom, V.: Confluence by decreasing diagrams: converted. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 306–320. Springer, Heidelberg (2008)
22. Yoshida, J., Aoto, T., Toyama, Y.: Automating confluence check of term rewriting systems. Computer Software (to appear) (in Japanese)
23. Zantema, H.: Termination of term rewriting: interpretation and type elimination. Journal of Symbolic Computation 17, 23–50 (1994)

A Proof Theoretic Analysis of Intruder Theories

Alwen Tiu and Rajeev Goré

Logic and Computation Group
College of Computer Science and Engineering
The Australian National University

Abstract. We consider the problem of intruder deduction in security protocol analysis: that is, deciding whether a given message M can be deduced from a set of messages Γ under the theory of blind signatures and arbitrary convergent equational theories modulo associativity and commutativity (AC) of certain binary operators. The traditional formulations of intruder deduction are usually given in natural-deduction-like systems and proving decidability requires significant effort in showing that the rules are "local" in some sense. By using the well-known translation between natural deduction and sequent calculus, we recast the intruder deduction problem as proof search in sequent calculus, in which locality is immediate. Using standard proof theoretic methods, such as permutability of rules and cut elimination, we show that the intruder deduction problem can be reduced, in polynomial time, to the elementary deduction problems, which amounts to solving certain equations in the underlying individual equational theories. We further show that this result extends to combinations of disjoint AC-convergent theories whereby the decidability of intruder deduction under the combined theory reduces to the decidability of elementary deduction in each constituent theory. Although various researchers have reported similar results for individual cases, our work shows that these results can be obtained using a systematic and uniform methodology based on the sequent calculus.

Keywords: AC convergent theories, sequent calculus, intruder deduction, security protocols.

1 Introduction

One of the fundamental aspects of the analysis of security protocols is the model of the intruder that seeks to compromise the protocols. In many situations, such a model can be described in terms of a deduction system which gives a formal account of the ability of the intruder to analyse and synthesize messages. As shown in many previous works (see, e.g., [2,6,7,9]), finding attacks on protocols can often be framed as the problem of deciding whether a certain formal expression is derivable in the deduction system which models the intruder capability. The latter is sometimes called the *intruder deduction problem*, or the (ground) reachability problem. A basic deductive account of the intruder's capability is based on the so-called Dolev-Yao model, which assumes perfect encryption. While this

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 103–117, 2009.
© Springer-Verlag Berlin Heidelberg 2009

model has been applied fruitfully to many situations, a stronger model of intruders is needed to discover certain types of attacks. A recent survey [11] shows that attacks on several protocols used in real-world communication networks can be found by exploiting algebraic properties of encryption functions.

The types of attacks mentioned in [11] have motivated many recent works in studying models of intruders in which the algebraic properties of the operators used in the protocols are taken into account [1,7,9,10,13,17]. In most of these, the intruder's capability is usually given as a natural-deduction-like deductive system. As is common in natural deduction, each constructor has a rule for introducing the constructor and one for eliminating the constructor. The elimination rule typically decomposes a term, reading the rule top-down: *e.g.*, a typical elimination rule for a pair $\langle M, N\rangle$ of terms is:

$$\frac{\Gamma \vdash \langle M, N\rangle}{\Gamma \vdash M}$$

Here, Γ denotes a set of terms, which represents the terms accumulated by the intruder over the course of its interaction with participants in a protocol. While a natural deduction formulation of deductive systems may seem "natural" and may reflect the meaning of the (logical) operators, it does not immediately give us a proof search strategy. Proof search means that we have to apply the rules bottom up, and as the above elimination rule demonstrates, this requires us to come up with a term N which might seem arbitrary. For a more complicated example, consider the following elimination rule for *blind signatures* [5,15,16].

$$\frac{\Gamma \vdash \mathsf{sign}(\mathsf{blind}(M, R), K) \quad \Gamma \vdash R}{\Gamma \vdash \mathsf{sign}(M, K)}$$

The basis for this rule is that the "unblinding" operation commutes with signature. Devising a proof search strategy in a natural deduction system containing this type of rule does not seem trivial. In most of the works mentioned above, in order to show the decidability results for the natural deduction system, one needs to prove that the system satisfies a notion of *locality*, i.e., in searching for a proof for $\Gamma \vdash M$, one needs only to consider expressions which are made of subterms from Γ and M. In addition, one has to also deal with the complication that arises from the use of the algebraic properties of certain operators.

In this work, we recast the intruder deduction problem as proof search in sequent calculus. A sequent calculus formulation of Dolev-Yao intruders was previously used by the first author in a formulation of open bisimulation for the spi-calculus [19] to prove certain results related to open bisimulation. The current work takes this idea further to include richer theories. Part of our motivation is to apply standard techniques, which have been well developed in the field of logic and proof theory, to the intruder deduction problem. In proof theory, sequent calculus is commonly considered a better calculus for studying proof search and decidability of logical systems, in comparison to natural deduction. This is partly due to the so-called "subformula" property (that is, the premise of every inference rule is made up of subterms of the conclusion of the

rule), which in most cases entails the decidability of the deductive system. It is therefore rather curious that none of the existing works on intruder deduction so far uses sequent calculus to structure proof search. We consider the ground intruder deduction problem (i.e., there are no variables in terms) under the class of *AC-convergent theories*. These are equational theories that can be turned into convergent rewrite systems, modulo associativity and commutativity of certain binary operators. Many important theories for intruder deduction fall into this category, e.g., theories for exclusive-or [7,9], Abelian groups [9], and more generally, certain classes of monoidal theories [10].

We show two main results. Firstly, we show that the decidability of intruder deduction under AC-convergent theories can be reduced, in polynomial time, to *elementary intruder deduction problems*, which involve only the equational theories under consideration. Secondly, we show that the intruder deduction problem for a combination of disjoint theories $E_1, \ldots, E_n$ can be reduced, in polynomial time, to the elementary deduction problem *for each theory* E_i. This means that if the elementary deduction problem is decidable for each E_i, then the intruder deduction problem under the combined theory is also decidable. We note that these decidability results are not really new, although there are slight differences and improvements over the existing works (see Section 7). Our contribution is more of a methodological nature. We arrive at these results using rather standard proof theoretical techniques, e.g., *cut-elimination* and permutability of inference rules, in a uniform and systematic way. In particular, we obtain locality of proof systems for intruder deduction, which is one of the main ingredients to decidability results in [9,7,13,12], for a wide range of theories that cover those studied in these works. Note that these works deal with a more difficult problem of deducibility constraints, which models *active intruders*, whereas we currently deal only with passive intruders. As future work, we plan to extend our approach to deal with active intruders.

The remainder of the paper is organised as follows. Section 2 presents two systems for intruder theories, one in natural deduction and the other in sequent calculus, and show that the two systems are equivalent. In Section 3, the sequent system is shown to enjoy cut-elimination. In Section 4, we show that cut-free sequent derivations can be transformed into a certain normal form. Using this result, we obtain another "linear" sequent system, from which the polynomial reducibility result follows. Section 5 discusses several example theories which can be found in the literature. Section 6 shows that the sequent system in Section 2 can be extended to cover any combination of disjoint AC-convergent theories, and the same decidability results also hold for this extension. Detailed proofs can be found in an extended version of the paper.[1]

2 Intruder Deduction Under AC Convergent Theories

We consider the following problem of formalising, given a set of messages Γ and a message M, whether M can be synthesized from the messages in Γ. We shall

[1] Available from `http://arxiv.org/abs/0804.0273`

write this judgment as $\Gamma \vdash M$. This is sometimes called the 'ground reachability' problem or the 'intruder deduction' problem in the literature.

Messages are formed from names, variables and function symbols. We shall assume the following sets: a countably infinite set N of names ranged over by a, b, c, d, m and n; a countably infinite set V of variables ranged over by x, y and z; and a finite set $\Sigma_C = \{\mathsf{pub}, \mathsf{sign}, \mathsf{blind}, \langle .,. \rangle, \{.\}_. \}$ of symbols representing the *constructors*. Thus pub is a public key constructor, sign is a public key encryption function, blind is the blinding encryption function (as in [15,16,5]), $\langle .,. \rangle$ is a pairing constructor, and $\{.\}_.$ is the Dolev-Yao symmetric encryption function. Additionally, we also assume a possibly empty equational theory E, whose signature is denoted with Σ_E. We require that $\Sigma_C \cap \Sigma_E = \emptyset$.[2] Function symbols (including constructors) are ranged over by f, g and h. The equational theory E may contain at most one associative-commutative function symbol, which we denote with $\oplus$, obeying the standard associative and commutative laws. We restrict ourselves to equational theories which can be represented by terminating and confluent rewrite systems, modulo the associativity and commutativity of $\oplus$. We consider the set of messages generated by the following grammar

$$M, N := a \mid x \mid \mathsf{pub}(M) \mid \mathsf{sign}(M, N) \mid \mathsf{blind}(M, N) \\ \mid \langle M, N \rangle \mid \{M\}_N \mid f(M_1, \ldots, M_k).$$

The message $\mathsf{pub}(M)$ denotes the public key generated from a private key M; $\mathsf{sign}(M, N)$ denotes a message M signed with a private key N; $\mathsf{blind}(M, N)$ denotes a message M encrypted with N using a special blinding encryption; $\langle M, N \rangle$ denotes a pair of messages; and $\{M\}_N$ denotes a message M encrypted with a key N using a Dolev-Yao symmetric encryption. The blinding encryption has a special property that it commutes with the sign operation, i.e., one can "unblind" a signed blinded message $\mathsf{sign}(\mathsf{blind}(M, r), k)$ using the blinding key r to obtain $\mathsf{sign}(M, k)$. This aspect of the blinding encryption is reflected in its elimination rules, as we shall see later. We denote with $V(M)$ the set of variables occurring in M. A term M is *ground* if $V(M) = \emptyset$. We shall be mostly concerned with ground terms, so unless stated otherwise, we assume implicitly that terms are ground (the only exception is Proposition 2 and Proposition 3).

We shall use several notions of equality so we distinguish them using the following notation: we shall write $M = N$ to denote syntactic equality, $M \equiv N$ to denote equality modulo associativity and commutativity (AC) of $\oplus$, and $M \approx_T N$ to denote equality modulo a given equational theory T. We shall sometimes omit the subscript in $\approx_T$ if it can be inferred from context.

Given an equational theory E, we denote with R_E the set of rewrite rules for E (modulo AC). We write $M \rightarrow_{R_E} N$ when M rewrites to N using one application of a rewrite rule in R_E. The definition of rewriting modulo AC is standard and is omitted here. The reflexive-transitive closure of $\rightarrow_{R_E}$ is denoted with $\rightarrow^*_{R_E}$. We shall often remove the subscript R_E when no confusion arises. A term M is in *E-normal form* if $M \not\rightarrow_{R_E} N$ for any N. We write $M{\downarrow_E}$ to denote the normal

[2] This restriction means that intruder theory such as homomorphic encryption is excluded. Nevertheless, it still covers a wide range of intruder theories.

form of M with respect to the rewrite system R_E, modulo commutativity and associativity of $\oplus$. Again, the index E is often omitted when it is clear which equational theory we refer to. This notation extends straightforwardly to sets, e.g., $\Gamma\downarrow$ denotes the set obtained by normalising all the elements of Γ. A term M is said to be *headed by* a symbol f if $M = f(M_1, \ldots, M_k)$. M is *guarded* if it is either a name, a variable, or a term headed by a constructor. A term M is an *E-alien term* if M is headed by a symbol $f \notin \Sigma_E$. It is a *pure E-term* if it contains only symbols from Σ_E, names and variables.

A *context* is a term with holes. We denote with $C^k[]$ a context with k-hole(s). When the number k is not important or can be inferred from context, we shall write $C[\ldots]$ instead. Viewing a context $C^k[]$ as a tree, each hole in the context occupies a unique position among the leaves of the tree. We say that a hole occurrence is the i-th hole of the context $C^k[]$ if it is the i-th hole encountered in an inorder traversal of the tree representing $C^k[]$. An E-context is a context formed using only the function symbols in Σ_E. We write $C[M_1, \ldots, M_k]$ to denote the term resulting from replacing the holes in the k-hole context $C^k[]$ with $M_1, \ldots, M_k$, with M_i occuping the i-th hole in $C^k[]$.

Natural deduction and sequent systems. The standard formulation of the judgment $\Gamma \vdash M$ is usually given in terms of a natural-deduction style inference system, as shown in Figure 1. We shall refer to this proof system as $\mathcal{N}$ and write $\Gamma \Vdash_{\mathcal{N}} M$ if $\Gamma \vdash M$ is derivable in $\mathcal{N}$. The deduction rules for Dolev-Yao encryption is standard and can be found in the literature, e.g., [6,9]. The blind signature rules are taken from the formulation given by Bernat and Comon-Lundh [5]. Note that the rule sign_E assumes implicitly that signing a message hides its contents. An alternative rule without this assumption would be

$$\frac{\Gamma \vdash \mathsf{sign}(M, K)}{\Gamma \vdash M}$$

The results of the paper also hold, with minor modifications, if we adopt this rule.

A sequent $\Gamma \vdash M$ is in *normal form* if M and all the terms in Γ are in normal form. Unless stated otherwise, in the following we assume that sequents are in normal form. The sequent system for intruder deduction, under the equational theory E, is given in Figure 2. We refer to this sequent system as $\mathcal{S}$ and write $\Gamma \Vdash_{\mathcal{S}} M$ to denote the fact that the sequent $\Gamma \vdash M$ is derivable in $\mathcal{S}$.

Unlike natural deduction rules, sequent rules also allow introduction of terms on the left hand side of the sequent. The rules p_L, e_L, sign_L, blind_{L1}, blind_{L2}, and gs are called *left introduction rules* (or simply *left rules*), and the rules $p_R, e_R, \mathsf{sign}_R, \mathsf{blind}_R$ are called *right introduction rules* (or simply, *right rules*). Notice that the rule gs is very similar to *cut*, except that we have the proviso that A is a subterm of a term in the lower sequent. This is sometimes called *analytic cut* in the proof theory literature. Analytic cuts are not problematic as far as proof search is concerned, since it still obeys the sub-formula property.

We need the rule gs because we do not have introduction rules for function symbols in Σ_E, in contrast to natural deduction. This rule is needed to "abstract"

$$\frac{M \in \Gamma}{\Gamma \vdash M}\ id \qquad \frac{\Gamma \vdash \{M\}_K \quad \Gamma \vdash K}{\Gamma \vdash M}\ e_E \qquad \frac{\Gamma \vdash M \quad \Gamma \vdash K}{\Gamma \vdash \{M\}_K}\ e_I$$

$$\frac{\Gamma \vdash \langle M, N\rangle}{\Gamma \vdash M}\ p_E \qquad \frac{\Gamma \vdash \langle M, N\rangle}{\Gamma \vdash N}\ p_E \qquad \frac{\Gamma \vdash M \quad \Gamma \vdash N}{\Gamma \vdash \langle M, N\rangle}\ p_I$$

$$\frac{\Gamma \vdash \mathsf{sign}(M, K) \quad \Gamma \vdash \mathsf{pub}(K)}{\Gamma \vdash M}\ \mathsf{sign}_E \qquad \frac{\Gamma \vdash M \quad \Gamma \vdash K}{\Gamma \vdash \mathsf{sign}(M, K)}\ \mathsf{sign}_I$$

$$\frac{\Gamma \vdash \mathsf{blind}(M, K) \quad \Gamma \vdash K}{\Gamma \vdash M}\ \mathsf{blind}_{E1} \qquad \frac{\Gamma \vdash M \quad \Gamma \vdash K}{\Gamma \vdash \mathsf{blind}(M, K)}\ \mathsf{blind}_I$$

$$\frac{\Gamma \vdash \mathsf{sign}(\mathsf{blind}(M, R), K) \quad \Gamma \vdash R}{\Gamma \vdash \mathsf{sign}(M, K)}\ \mathsf{blind}_{E2}$$

$$\frac{\Gamma \vdash M_1 \quad \cdots \quad \Gamma \vdash M_n}{\Gamma \vdash f(M_1, \ldots, M_n)}\ f_I, \text{ where } f \in \Sigma_E \qquad \frac{\Gamma \vdash N}{\Gamma \vdash M}\ \approx, \text{ where } M \approx_E N$$

Fig. 1. System $\mathcal{N}$: a natural deduction system for intruder deduction

E-alien subterms in a sequent (in the sense of the variable abstraction technique common in unification theory, see e.g., [18,4]), which is needed to prove that the cut rule is redundant. For example, let E be a theory containing only the associativity and the commutativity axioms for $\oplus$. Then the sequent $a, b \vdash \langle a, b\rangle \oplus a$ should be provable without cut. Apart from the gs rule, the only other way to prove this is by using the id rule. However, id is not applicable, since no E-context $C[...]$ can obey $C[a, b] \approx \langle a, b\rangle \oplus a$ because E-contexts can contain only symbols from Σ_E and thus cannot contain $\langle ., .\rangle$. Therefore we need to "abstract" the term $\langle a, b\rangle$ in the right hand side, via the gs rule:

$$\dfrac{\dfrac{\dfrac{}{a, b \vdash a}\ id \quad \dfrac{}{a, b \vdash b}\ id}{a, b \vdash \langle a, b\rangle}\ p_R \qquad \dfrac{}{a, b, \langle a, b\rangle \vdash \langle a, b\rangle \oplus a}\ id}{a, b \vdash \langle a, b\rangle \oplus a}\ gs$$

The third id rule instance (from the left) is valid because we have $C[\langle a, b\rangle, a] \equiv \langle a, b\rangle \oplus a$, where $C[., .] = [.] \oplus [.]$.

Provability in the natural deduction system and in the sequent system are related via the standard translation, i.e., right rules in sequent calculus correspond to introduction rules in natural deduction and left rules corresponds to elimination rules. The straightforward translation from natural deduction to sequent calculus uses the cut rule.

Proposition 1. *The judgment $\Gamma \vdash M$ is provable in the natural deduction system $\mathcal{N}$ if and only if $\Gamma\downarrow \vdash M\downarrow$ is provable in the sequent system $\mathcal{S}$.*

3 Cut Elimination for $\mathcal{S}$

We now show that the cut rule is redundant for $\mathcal{S}$.

$$\frac{M \approx_E C[M_1, \ldots, M_k] \quad C[\,] \text{ an } E\text{-context, and } M_1, \ldots, M_k \in \Gamma}{\Gamma \vdash M}\ id \qquad \frac{\Gamma \vdash M \quad \Gamma, M \vdash T}{\Gamma \vdash T}\ cut$$

$$\frac{\Gamma, \langle M, N\rangle, M, N \vdash T}{\Gamma, \langle M, N\rangle \vdash T}\ p_L \qquad \frac{\Gamma \vdash M \quad \Gamma \vdash N}{\Gamma \vdash \langle M, N\rangle}\ p_R$$

$$\frac{\Gamma, \{M\}_K \vdash K \quad \Gamma, \{M\}_K, M, K \vdash N}{\Gamma, \{M\}_K \vdash N}\ e_L \qquad \frac{\Gamma \vdash M \quad \Gamma \vdash K}{\Gamma \vdash \{M\}_K}\ e_R$$

$$\frac{\Gamma, \mathsf{sign}(M, K), \mathsf{pub}(L), M \vdash N}{\Gamma, \mathsf{sign}(M, K), \mathsf{pub}(L) \vdash N}\ \mathsf{sign}_L, K \equiv L \qquad \frac{\Gamma \vdash M \quad \Gamma \vdash K}{\Gamma \vdash \mathsf{sign}(M, K)}\ \mathsf{sign}_R$$

$$\frac{\Gamma, \mathsf{blind}(M, K) \vdash K \quad \Gamma, \mathsf{blind}(M, K), M, K \vdash N}{\Gamma, \mathsf{blind}(M, K) \vdash N}\ \mathsf{blind}_{L1} \qquad \frac{\Gamma \vdash M \quad \Gamma \vdash K}{\Gamma \vdash \mathsf{blind}(M, K)}\ \mathsf{blind}_R$$

$$\frac{\Gamma, \mathsf{sign}(\mathsf{blind}(M, R), K) \vdash R \quad \Gamma, \mathsf{sign}(\mathsf{blind}(M, R), K), \mathsf{sign}(M, K), R \vdash N}{\Gamma, \mathsf{sign}(\mathsf{blind}(M, R), K) \vdash N}\ \mathsf{blind}_{L2}$$

$$\frac{\Gamma \vdash A \quad \Gamma, A \vdash M}{\Gamma \vdash M}\ gs, A \text{ is a guarded subterm of } \Gamma \cup \{M\}$$

Fig. 2. System $\mathcal{S}$: a sequent system for intruder deduction

Definition 1. *An inference rule R in a proof system $\mathcal{D}$ is* admissible for $\mathcal{D}$ *if for every sequent $\Gamma \vdash M$ derivable in $\mathcal{D}$, there is a derivation of the same sequent in $\mathcal{D}$ without instances of R.*

The *cut-elimination* theorem for $\mathcal{S}$ states that the cut rule is admissible for $\mathcal{S}$. Before we proceed with the main cut elimination proof, we first prove a basic property of equational theories and rewrite systems, which is concerned with a technique called *variable abstraction* [18,4].

Given derivation Π, the *height* of the derivation, denoted by $|\Pi|$, is the length of a longest branch in Π. Given a normal term M, the *size* $|M|$ of M is the number of function symbols, names and variables appearing in M.

In the following, we consider slightly more general equational theories than in the previous section: each AC theory E can be a theory obtained from a disjoint combination of AC theories $E_1, \ldots, E_k$, where each E_i has at most one AC operator $\oplus_i$. This allows us to reuse the results for a more general case later.

Definition 2. *Let E be a disjoint combination of AC convergent theories E_1, $\ldots$, E_n. A term M is a* quasi-E_i term *if every E_i-alien subterm of M is in E-normal form.*

For example, let $E = \{h(x, x) \approx x\}$. Then $h(\langle a, b\rangle, c)$ is a quasi E-term, whereas $h(\langle a, b\rangle, \langle h(a, a), b\rangle)$ is not, since its E-alien subterm $\langle h(a, a), b\rangle$ is not in its E-normal form $\langle a, b\rangle$. Obviously, any E normal term is a quasi E_i term.

In the following, given an equational theory E, we assume the existence of a function v_E, which assigns a variable from V to each ground term such that

$v_E(M) = v_E(N)$ if and only if $M \approx_E N$. In other words, v_E assigns a unique variable to each equivalence class of ground terms induced by $\approx_E$.

Definition 3. *Let E be an equational theory obtained by disjoint combination of AC theories $E_1, \ldots, E_n$. The E_i* abstraction function *F_{E_i} is a function mapping ground terms to pure E_i terms, defined recursively as follows:*

$$F_{E_i}(u) = \begin{cases} u, & \text{if } u \text{ is a name,} \\ f(F_{E_i}(u_1), \ldots, F_{E_i}(u_k)), & \text{if } u = f(u_1, \ldots, u_k) \text{ and } f \in \Sigma_{E_i}, \\ v_E(u), & \text{otherwise.} \end{cases}$$

It can be easily shown that the function F_{E_i} preserves the equivalence relation $\equiv$. That is, if $M \equiv N$ then $F_{E_i}(M) \equiv F_{E_i}(N)$.

Proposition 2. *Let E be a disjoint combination of $E_1, \ldots, E_n$. If M is a quasi E_i term and $M \rightarrow^*_{R_E} N$, then N is a quasi E_i term and $F_{E_i}(M) \rightarrow^*_{R_E} F_{E_i}(N)$.*

Proposition 3. *Let E be a disjoint combination of $E_1, \ldots, E_n$. If M and N are quasi E_i terms and $F_{E_i}(M) \rightarrow^*_{R_E} F_{E_i}(N)$, then $M \rightarrow^*_{R_E} N$.*

We now show some important proof transformations needed to prove cut elimination, i.e., in an inductive argument to reduce the size of cut terms. In the following, when we write that a sequent $\Gamma \vdash M$ is derivable, we mean that it is derivable in the proof system $\mathcal{S}$, with a fixed AC theory E.

Lemma 1. *Let Π be a derivation of $M_1, \ldots, M_k \vdash N$. Then for any $M'_1, \ldots, M'_k$ and N' such that $M_i \equiv M'_i$ and $N \equiv N'$, there is a derivation Π' of $M'_1, \ldots, M'_k \vdash N'$ such that $|\Pi| = |\Pi'|$.*

Lemma 2. *Let X and Y be terms in normal form and let f be a binary constructor. If $\Gamma, f(X,Y) \vdash M$ is cut-free derivable, then so is $\Gamma, X, Y \vdash M$.*

The more interesting case in the proof of Lemma 2 is when $\Gamma, f(X,Y) \vdash M$ is proved by an application of the *id* rule where $f(X,Y)$ is active. That is, we have $C[f(X,Y), M_1, \ldots, M_k] \approx_E M$, where $M_1, \ldots, M_k \in \Gamma$, for some E-context $C[..]$. Since M is in normal form, we have

$$C[f(X,Y), M_1, \ldots, M_k] \rightarrow^* M. \tag{1}$$

There are two cases to consider in the construction of a proof for $\Gamma, X, Y \vdash M$. If $f(X,Y)$ occurs as a subterm of M or Γ, then we simply apply the *gs* rule (bottom up) to abstract the term $f(X,Y)$ and then apply the *id* rule. Otherwise, we use the variable abstraction techniques (Proposition 2 and Proposition 3) to abstract $f(X,Y)$ from the rewrite steps (1) above, and then replace its abstraction with X to obtain: $C[X, M_1, \ldots, M_k] \rightarrow^* M$. That is, the *id* rule is applicable to the sequent $\Gamma, X, Y \vdash M$, with X taking the role of $f(X,Y)$.

Lemma 3. *Let $X_1, \ldots, X_k$ be normal terms and let Π be a cut-free derivation of $\Gamma, f(X_1, \ldots, X_k)\downarrow \vdash M$, where $f \in \Sigma_E$. Then there exists a cut-free derivation Π' of $\Gamma, X_1, \ldots, X_k \vdash M$.*

Lemma 4. *Let $M_1, \ldots, M_k$ be terms in normal form and let $C[\ldots]$ be a k-hole E-context. If $\Gamma, C[M_1, \ldots, M_k]\downarrow \vdash M$ is cut-free derivable, then so is $\Gamma, M_1, \ldots, M_k \vdash M$.*

One peculiar aspect of the sequent system $\mathcal{S}$ is that in the introduction rules for encryption functions (including blind signatures), there is no switch of polarities for the encryption key. For example, in the introduction rule for $\{M\}_K$, both on the left and on the right, the key K appears on the right hand side of a premise of the rule. This means that there is no exchange of information between the left and the right hand side of sequents, unlike, say, typical implication rules in logic. This gives rise to an easy cut elimination proof, where we need only to measure the complexity of the left premise of a cut in determining the cut rank.

Theorem 1. *The cut rule is admissible for $\mathcal{S}$.*

4 Normal Derivations and Decidability

We now turn to the question of the decidability of the deduction problem $\Gamma \vdash M$. This problem is known already for several AC theories, e.g., exclusive-or, abelian groups and their extensions with a homomorphism axiom [1,7,9,12,13]. What we would like to show here is how the decidability result can be reduced to a more elementary decision problem, defined as follows.

Definition 4. *Given an equational theory E, the* elementary deduction problem *for E, written $\Gamma \Vdash_E M$, is the problem of deciding whether the id rule is applicable to the sequent $\Gamma \vdash M$ (by checking whether there exists an E-context $C[\ldots]$ and terms $M_1, \ldots, M_k \in \Gamma$ such that $C[M_1, \ldots, M_k] \approx_E M$).*

Note that as a consequence of Proposition 2 and Proposition 3, in checking elementary deducibility, it is enough to consider the pure E equational problem where all E-alien subterms are abstracted, i.e., we have

$$C[M_1, \ldots, M_k] \approx_E M \qquad \text{iff} \qquad C[F_E(M_1), \ldots, F_E(M_k)] \approx_E F_E(M).$$

Our notion of elementary deduction corresponds roughly to the notion of "recipe" in [1], but we note that the notion of a recipe is a stronger one, since it bounds the size of the equational context.

The cut free sequent system does not strictly speaking enjoy the "sub-formula" property, i.e., in blind_{L2}, the premise sequent has a term which is not a subterm of any term in the lower sequent. However, it is easy to see that, reading the rules bottom up, we only ever introduce terms which are smaller than the terms in the lower sequent. Thus a naive proof search strategy which non-deterministically tries all applicable rules and avoids repeated sequents will eventually terminate. This procedure is of course rather expensive. We show that we can obtain a better complexity result by analysing the structures of cut-free derivations. Recall that the rules $p_L, e_L, \mathsf{sign}_L, \mathsf{blind}_{L1}, \mathsf{blind}_{L2}$ and gs are called left rules (the other rules are right rules).

$$\frac{\Gamma \Vdash_{\mathcal{R}} M}{\Gamma \vdash M}\ r \qquad \frac{\Gamma, \{M\}_K, M, K \vdash N}{\Gamma, \{M\}_K \vdash N}\ le, \text{ where } \Gamma, \{M\}_K \Vdash_{\mathcal{R}} K$$

$$\frac{\Gamma, \langle M, N\rangle, M, N \vdash T}{\Gamma, \langle M, N\rangle \vdash T}\ lp \qquad \frac{\Gamma, \mathsf{sign}(M, K), \mathsf{pub}(L), M \vdash N}{\Gamma, \mathsf{sign}(M, K), \mathsf{pub}(L) \vdash N}\ \mathsf{sign}, K \equiv L$$

$$\frac{\Gamma, \mathsf{blind}(M, K), M, K \vdash N}{\Gamma, \mathsf{blind}(M, K) \vdash N}\ \mathsf{blind}_1, \text{ where } \Gamma, \mathsf{blind}(M, K) \Vdash_{\mathcal{R}} K$$

$$\frac{\Gamma, \mathsf{sign}(\mathsf{blind}(M, R), K), \mathsf{sign}(M, K), R \vdash N}{\Gamma, \mathsf{sign}(\mathsf{blind}(M, R), K) \vdash N}\ \mathsf{blind}_2,$$

where $\Gamma, \mathsf{sign}(\mathsf{blind}(M, R), K) \Vdash_{\mathcal{R}} R$.

$$\frac{\Gamma, A \vdash M}{\Gamma \vdash M}\ ls, \text{where } A \text{ is a guarded subterm of } \Gamma \cup \{M\} \text{ and } \Gamma \Vdash_{\mathcal{R}} A.$$

Fig. 3. System $\mathcal{L}$: a linear proof system for intruder deduction

Definition 5. *A cut-free derivation Π is said to be a* normal derivation *if it satisfies the following conditions:*

1. *no left rule appears above a right rule;*
2. *no left rule appears immediately above the left-premise of a branching left rule (i.e., all left rules except p_L and sign_L).*

Proposition 4. *If $\Gamma \vdash M$ is derivable then it has a normal derivation.*

In a normal derivation, the left branch of a branching left rule is provable using only right rules and *id*. This means that we can represent a normal derivation as a sequence (reading the proof bottom-up) of sequents, each of which is obtained from the previous one by adding terms composed of subterms of the previous sequent, with the proviso that certain subterms can be constructed using right-rules. Let us denote with $\Gamma \Vdash_{\mathcal{R}} M$ the fact that the sequent $\Gamma \vdash M$ is provable using only the right rules and *id*. This suggests a more compact deduction system for intruder deduction, called system $\mathcal{L}$, given in Figure 3.

Proposition 5. *Every sequent $\Gamma \vdash M$ is provable in $\mathcal{S}$ if and only if it is provable in $\mathcal{L}$.*

We now show that the decidability of the deduction problem $\Gamma \Vdash_{\mathcal{S}} M$ can be reduced to decidability of elementary deduction problems. We consider a representation of terms as directed acyclic graphs (DAG), with maximum sharing of subterms. Such a representation is quite standard and can be found in, e.g., [1], so we will not go into the details here.

In the following, we denote with $st(\Gamma)$ the set of subterms of the terms in Γ. In the DAG representation of Γ, the number of distinct nodes in the DAG representing distinct subterms of Γ co-incides with the cardinality of $st(\Gamma)$. A

term M is a *proper subterm* of N if M is a subterm of N and $M \neq N$. We denote with $pst(\Gamma)$ the set of proper subterms of Γ, and we define

$$sst(\Gamma) = \{\mathsf{sign}(M, N) \mid M, N \in pst(\Gamma)\}.$$

The *saturated set* of Γ, written $St(\Gamma)$, is the set

$$St(\Gamma) = \Gamma \cup pst(\Gamma) \cup sst(\Gamma).$$

The cardinality of $St(\Gamma)$ is at most quadratic in the size of $st(\Gamma)$. If Γ is represented as a DAG, one can compute the DAG representation of $St(\Gamma)$ in polynomial time, with only a quadratic increase of the size of the graph. Given a DAG representation of $St(\Gamma \cup \{M\})$, we can represent a sequent $\Gamma \vdash M$ by associating each node in the DAG with a tag which indicates whether or not the term represented by the subgraph rooted at that node appears in Γ or M. Therefore, in the following complexity results for deducibility problem $\Gamma \Vdash_S M$ (for some proof system S), we assume that the input consists of the DAG representation of the saturated set $St(\Gamma \cup \{M\})$, together with approriate tags in the nodes. Since each tag takes only a fixed amount of space (e.g., a two-bit data structure should suffice), we shall state the complexity result w.r.t. the size of $St(\Gamma \cup \{M\})$.

Definition 6. *Let $\Gamma \Vdash_{\mathcal{D}} M$ be a deduction problem, where $\mathcal{D}$ is some proof system, and let n be the size of $St(\Gamma \cup \{M\})$. Let E be the equational theory associated with $\mathcal{D}$. Suppose that the elementary deduction problem in E has complexity $O(f(m))$, where m is the size of the input. Then the problem $\Gamma \Vdash_{\mathcal{D}} M$ is said to be* polynomially reducible *to the elementary deduction problem $\Vdash_E$ if it has complexity $O(n^k \times f(n))$ for some constant k.*

A key lemma in proving the decidability result is the following invariant property of linear proofs.

Lemma 5. *Let Π be an $\mathcal{L}$-derivation of $\Gamma \vdash M$. Then for every sequent $\Gamma' \vdash M'$ occurring in Π, we have $\Gamma' \cup \{M'\} \subseteq St(\Gamma \cup \{M\})$.*

The existence of linear size proofs then follows from the above lemma.

Lemma 6. *If there is an $\mathcal{L}$-derivation of $\Gamma \vdash M$ then there is an $\mathcal{L}$-derivation of the same sequent whose length is at most $|St(\Gamma \cup \{M\})|$.*

Another useful observation is that the left-rules of $\mathcal{L}$ are *invertible*; at any point in proof search, we do not lose provability by applying any left rule. Polynomial reducibility of $\Vdash_{\mathcal{L}}$ to $\Vdash_E$ can then be proved by a deterministic proof search strategy which systematically tries all applicable rules.

Theorem 2. *The decidability of the relation $\Vdash_{\mathcal{L}}$ is polynomially reducible to the decidability of elementary deduction $\Vdash_E$.*

Note that in the case where the theory E is empty, we obtain a PTIME decision procedure for intruder deduction with blind signatures.

5 Some Example Theories

We now consider several concrete AC convergent theories that are often used in reasoning about security protocols. Decidability of intruder deduction under these theories has been extensively studied [1,7,9,10,13,17]. These results can be broadly categorized into those with explicit pairing and encryption constructors, e.g., [9,17], and those where the constructors are part of the equational theories, e.g., [1,10]. For the latter, one needs explicit decryption operators with, e.g., an equation like $dec(\{M\}_N, N) \approx M$. Decidability results for these deduction problems are often obtained by separating elementary deducibility from the general deduction problem. This is obtained by studying some form of normal derivations in a natural deduction setting. Such a reduction, as has been shown in the previous section, applies to our calculus in a more systematic fashion.

Exclusive-or. The signature of this theory consists of a binary operator $\oplus$ and a constant 0. The theory is given by the axioms of associativity and commutativity of $\oplus$ together with the axiom $x \oplus x \approx 0$ and $x \oplus 0 \approx x$. This theory can be turned into an AC convergent rewrite system with the following rewrite rules:

$$x \oplus x \to 0 \qquad \text{and} \qquad x \oplus 0 \to x.$$

Checking $\Gamma \Vdash_E M$ can be done in PTIME, as shown in, e.g., [7].

Abelian groups. The exclusive-or theory is an instance of Abelian groups, where the inverse of an element is the element itself. The more general case of Abelian groups includes an inverse operator, denoted with I here. The equality theory for Abelian groups is given by the axioms of associativity and commutativity, plus the theory $\{x \oplus 0 \approx 0, x \oplus I(x) \approx 0\}$. The equality theory of Abelian groups can be turned into a rewrite system modulo AC by orienting the above equalities from left to right, in addition to the following rewrite rules:

$$I(x \oplus y) \to I(x) \oplus I(y) \qquad I(I(x)) \to x \qquad I(0) \to 0.$$

One can also obtain an AC convergent rewrite system for an extension of Abelian groups with a homomorphism axiom involving a unary operator h: $h(x \oplus y) = h(x) \oplus h(y)$. In this case, the rewrite rules above need to be extended with

$$h(x \oplus y) \to h(x) \oplus h(y) \qquad h(0) \to 0 \qquad h(I(x)) \to I(h(x)).$$

Decidability of elementary deduction under Abelian groups (with homomorphism) can be reduced to solving a system of linear equations over some semirings (see [12] for details).

6 Combining Disjoint Convergent Theories

We now consider the intruder deduction problem under a convergent AC theory E, which is obtained from the union of pairwise disjoint convergent AC theories

$E_1, \ldots, E_n$. Each theory E_i may contain an associative-commutative binary operator, which we denote with $\oplus_i$. We show that the intruder deduction problem under E can be reduced to the elementary deduction problem of each E_i.

Given a term $M = f(M_1, \ldots, M_k)$, where f is a function symbol (i.e., a constructor, an equational symbol or $\oplus$), the terms $M_1, \ldots, M_k$ are called the *immediate subterms* of M. Given a term M and a subterm occurrence N in M, we say that N is a *cross-theory subterm* of M if N is headed with a symbol $f \in \Sigma_{E_i}$ and it is an immediate subterm of a subterm in M which is headed by a symbol $g \in \Sigma_{E_j}$, where $i \neq j$. We shall also refer to N as an E_{ij}*-subterm* of M when we need to be explicit about the equational theories involved.

Throughout this section, we consider a sequent system $\mathcal{D}$, whose rules are those of $\mathcal{S}$, but with *id* replaced by the rule below left and with the addition of the rule below right where N is a cross-theory subterm of some term in $\Gamma \cup \{M\}$:

$$\frac{\begin{array}{c} M \approx_E C[M_1, \ldots, M_k] \\ C[\,] \text{ an } E_i\text{-context, and } M_1, \ldots, M_k \in \Gamma \end{array}}{\Gamma \vdash M}\ id_{E_i} \qquad \frac{\Gamma \vdash N \quad \Gamma, N \vdash M}{\Gamma \vdash M}\ cs$$

The analog of Proposition 1 holds for $\mathcal{D}$. Its proof is a straightforward adaptation of the proof of Proposition 1.

Proposition 6. *The judgment $\Gamma \vdash M$ is provable in the natural deduction system $\mathcal{N}$, under theory E, if and only if $\Gamma\downarrow \vdash M\downarrow$ is provable in the sequent system $\mathcal{D}$.*

Cut elimination also holds for $\mathcal{D}$. Its proof is basically the same as the proof for $\mathcal{S}$, since the "logical structures" (i.e., those concerning constructors) are the same. The only difference is in the treatment of abstracted terms (the rules *gs* and *cs*). In $\mathcal{D}$ we allow abstraction of arbitrary cross-theory subterms, in addition to guarded subterm abstraction. The crucial part of the proof in this case relies on the variable abstraction technique (Proposition 2 and Proposition 3), which applies to both guarded subterm and cross-theory subterm abstraction.

Theorem 3. *The cut rule is admissible for $\mathcal{D}$.*

The decidability result for $\mathcal{S}$ also holds for $\mathcal{D}$. This can be proved with straightforward modifications of the similar proof for $\mathcal{S}$, since the extra rule *cs* has the same structure as *gs* in $\mathcal{S}$. It is easy to see that the same normal forms for $\mathcal{S}$ also holds for $\mathcal{D}$, with *cs* considered as a left-rule. It then remains to design a linear proof system for $\mathcal{D}$. We first define the notion of right-deducibility: The relation $\Gamma \Vdash_{\mathcal{RD}} M$ holds if and only if the sequent $\Gamma \vdash M$ is derivable in $\mathcal{D}$ using only the right rules. We next define a linear system for $\mathcal{D}$, called $\mathcal{LD}$, which consists of the rules of $\mathcal{L}$ defined in the previous section, but with the proviso $\Gamma \Vdash_{\mathcal{R}} M$ changed to $\Gamma \Vdash_{\mathcal{RD}} M$, and with the additional rule:

$$\frac{\Gamma, R \vdash M}{\Gamma \vdash M}\ lcs$$

where R is a cross-theory subterm of some term in $\Gamma \cup \{M\}$ and $\Gamma \Vdash_{\mathcal{RD}} R$.

Proposition 7. *A sequent $\Gamma \vdash M$ is provable in $\mathcal{D}$ if and only if it is provable in $\mathcal{LD}$.*

The notion of polynomial reducibility is slightly changed. Suppose each elementary deduction problem in E_i is bounded by $O(f(m))$. Let m be the size of $St(\Gamma \cup \{M\})$. Then the deduction problem $\Gamma \Vdash_{\mathcal{D}} M$ is polynomially reducible to $\Vdash_{E_1}, \ldots, \Vdash_{E_n}$ if it has complexity $O(m^k f(m))$, for some constant k.

Theorem 4. *The decidability of the relation $\Vdash_{\mathcal{LD}}$ is polynomially reducible to the decidability of elementary deductions $\Vdash_{E_1}, \ldots, \Vdash_{E_n}$.*

7 Conclusion and Related Work

We have shown that decidability of the intruder deduction problem, under a range of equational theories, can be reduced to the simpler problem of elementary deduction, which amounts to solving equations in the underlying equational theories. This reduction is obtained in a purely proof theoretical way, using standard techniques such as cut elimination and permutation of inference rules.

There are several existing works in the literature that deal with intruder deduction. Our work is more closely related to, e.g., [9,12,17], in that we do not have explicit destructors (projection, decryption, unblinding), than, say, [1,10]. In the latter work, these destructors are considered part of the equational theory, so in this sense our work slightly extends theirs to allow combinations of explicit and implicit destructors. A drawback for the approach with explicit destructors is that one needs to consider these destructors together with other algebraic properties in proving decidability, although recent work in combining decidable theories [3] allows one to deal with them modularly. Combination of intruder theories has been considered in [8,3,14], as part of their solution to a more difficult problem of deducibility constraints which assumes active intruders. In particular, Delaune, et. al., [14] obtain results similar to what we have here concerning combination of AC theories. One difference between these works and ours is in how this combination is derived. Their approach is more algorithmic whereas our result is obtained through analysis of proof systems.

It remains to be seen whether sequent calculus, and its associated proof techniques, can prove useful for richer theories. For certain deduction problems, i.e., those in which the constructors interact with the equational theory, there does not seem to be general results like the ones we obtain for theories with no interaction with the constructors. One natural problem where this interaction occurs is the theory with homomorphic encryption, e.g., like the one considered in [17]. Another interesting challenge is to see how sequent calculus can be used to study the more difficult problem of solving intruder deduction constraints, e.g., like those studied in [9,7,13].

Acknowledgement. We thank the anonymous referees of earlier drafts of this paper for their careful reading and helpful comments. This work has been supported by the Australian Research Council (ARC) Discovery Project DP0880549.

References

1. Abadi, M., Cortier, V.: Deciding knowledge in security protocols under equational theories. Theor. Comput. Sci. 367(1-2), 2–32 (2006)
2. Amadio, R.M., Lugiez, D.: On the reachability problem in cryptographic protocols. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 380–394. Springer, Heidelberg (2000)
3. Arnaud, M., Cortier, V., Delaune, S.: Combining algorithms for deciding knowledge in security protocols. In: Konev, B., Wolter, F. (eds.) FroCos 2007. LNCS, vol. 4720, pp. 103–117. Springer, Heidelberg (2007)
4. Baader, F., Schulz, K.U.: Unification in the union of disjoint equational theories: Combining decision procedures. J. Sym. Comp. 21(2), 211–243 (1996)
5. Bernat, V., Comon-Lundh, H.: Normal proofs in intruder theories. In: Okada, M., Satoh, I. (eds.) ASIAN 2006. LNCS, vol. 4435, pp. 151–166. Springer, Heidelberg (2008)
6. Boreale, M.: Symbolic trace analysis of cryptographic protocols. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 667–681. Springer, Heidelberg (2001)
7. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: An NP decision procedure for protocol insecurity with xor. In: LICS, pp. 261–270 (2003)
8. Chevalier, Y., Rusinowitch, M.: Combining intruder theories. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 639–651. Springer, Heidelberg (2005)
9. Comon-Lundh, H., Shmatikov, V.: Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In: LICS, pp. 271–280. IEEE Computer Society Press, Los Alamitos (2003)
10. Cortier, V., Delaune, S.: Deciding knowledge in security protocols for monoidal equational theories. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS, vol. 4790, pp. 196–210. Springer, Heidelberg (2007)
11. Cortier, V., Delaune, S., Lafourcade, P.: A survey of algebraic properties used in cryptographic protocols. Journal of Computer Security 14(1), 1–43 (2006)
12. Delaune, S.: Easy intruder deduction problems with homomorphisms. Inf. Process. Lett. 97(6), 213–218 (2006)
13. Delaune, S., Lafourcade, P., Lugiez, D., Treinen, R.: Symbolic protocol analysis in presence of a homomorphism operator and *exclusive or*. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 132–143. Springer, Heidelberg (2006)
14. Delaune, S., Lafourcade, P., Lugiez, D., Treinen, R.: Symbolic protocol analysis for monoidal equational theories. Inf. Comput. 206(2-4), 312–351 (2008)
15. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: Seberry, J., Zheng, Y. (eds.) AUSCRYPT 1992. LNCS, vol. 718, pp. 244–251. Springer, Heidelberg (1993)
16. Kremer, S., Ryan, M.: Analysis of an electronic voting protocol in the applied pi calculus. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 186–200. Springer, Heidelberg (2005)
17. Lafourcade, P., Lugiez, D., Treinen, R.: Intruder deduction for the equational theory of abelian groups with distributive encryption. Inf. Comput. 205(4), 581–623 (2007)
18. Schmidt-Schauß, M.: Unification in a combination of arbitrary disjoint equational theories. J. Symb. Comput. 8(1/2), 51–99 (1989)
19. Tiu, A.: A trace based bisimulation for the spi calculus: An extended abstract. In: Shao, Z. (ed.) APLAS 2007. LNCS, vol. 4807, pp. 367–382. Springer, Heidelberg (2007)

Flat and One-Variable Clauses for Single Blind Copying Protocols: The XOR Case

Helmut Seidl and Kumar Neeraj Verma

Technische Universität München
Institut für Informatik, 85748 Garching, Germany

Abstract. In cryptographic protocols with the single blind copying restriction, at most one piece of unknown data is allowed to be copied in each step of the protocol. The secrecy problem for such protocols can be modeled as the satisfiability problem for the class of first-order Horn clauses called flat and one-variable Horn clauses, and is known to be DEXPTIME-complete. We show that when an XOR operator is additionally present, then the secrecy problem is decidable in 3-EXPTIME. We also note that replacing XOR by the theory of associativity-commutativity or by the theory of Abelian groups, or removing some of the syntactic restrictions on the clauses, leads to undecidability.

1 Introduction

Cryptographic protocols are rules for exchange of messages between participants over an insecure network, and are meant to ensure security objectives like secrecy, authentication, etc. The messages exchanged are built from basic data like identities, public and private keys, and random numbers, by applying cryptographic operations like encryption, decryption and hashing. Because of the difficulty of detecting subtle flaws in even simple looking protocols, automated verification of such protocols has received considerable interest. This is commonly done under the well-known Dolev-Yao model [12] which in particular includes the *perfect-cryptography* assumption, which treats cryptographic operations as black boxes, thus allowing a symbolic analysis of protocols.

Since the general verification problem is undecidable, many researchers have tried to obtain decidability results by identifying reasonable conditions on the protocols. One common approach is to bound the number of allowed sessions of the protocol. In this case the secrecy problem becomes co-NP-complete [15]. This however helps us to detect only some of the security flaws. To certify a protocol, we need to consider unbounded number of possibly interleaved executions of the protocol, possibly with other *safe* abstractions, i.e. those which do not miss any flaws. A common safe abstraction is to allow only a bounded number of nonces, i.e. random numbers, to be used in infinitely many sessions. The secrecy problem still remains undecidable, though becomes decidable if message size is also bounded [13].

Instead of bounding message size, Comon-Lundh and Cortier [5,8] introduced the *single blind copying* restriction. Intuitively this restriction means that in each step involving receiving and sending of a message by a participant, at most one piece of data is blindly copied by the participant. This is a restriction often satisfied by protocols in the

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 118–132, 2009.
© Springer-Verlag Berlin Heidelberg 2009

literature. Such protocols are conveniently modeled by first-order Horn clauses called *flat and one-variable clauses*. Comon-Lundh and Cortier showed that this class can be decided in 3-EXPTIME. We further showed in previous work [16] that the problem is in fact DEXPTIME-complete.

Another line of work is a partial weakening of the perfect cryptography assumption, by taking into account algebraic properties of operations. Examples of such operations are exclusive-or (XOR) and modular exponentiation. Other properties of operations which occur commonly are associativity-commutativity and the Abelian group laws. On the one hand, such properties sometimes are necessary for the correct behavior of protocols, and on the other hand, certain flaws can be missed if our analysis ignores these properties. A detailed survey of such protocols and decision results about them can be found in [10]. Under the bounded sessions assumption, the verification problem has been shown to be decidable in presence of various operations like XOR and modular exponentiation [2,3,7]. In the case of unbounded number of sessions, we run into undecidability issues as in the case without algebraic properties. In this paper we show however that if the single blind copying protocols considered above are further allowed to use an XOR operator, then the secrecy problem is decidable in 3-EXPTIME.

The paper is organized as follows. We define our class of protocols as well as flat and one-variable clauses in Section 2. Various related classes as well as undecidability results are discussed in Section 2.2. To present our decision procedure we start in Section 3 by presenting some useful results on unification in presence of the XOR theory. Our main algorithm and its analysis are then presented in Section 4.

2 Modeling and Some Undecidability Results

We fix an infinite set $\mathsf{X} = \{\mathsf{x}_1, \mathsf{x}_2, \ldots\}$ of variables and a signature $\Gamma = \{\mathsf{f}, \mathsf{g}, \ldots\}$ of finitely many function symbols, each with a fixed arity. We assume Γ to contain the binary symbol $\oplus$, representing the XOR operator, and the nullary symbol $\mathsf{0}$, representing the unit of this binary operator. Terms built over these variables and symbols are defined as usual. Recall that a congruence relation $\equiv$ is an equivalence relation such that $f(s_1, \ldots, s_i, s, s_{i+1}, \ldots, s_n) \equiv f(s_1, \ldots, s_i, t, s_{i+1}, \ldots, s_n)$ whenever $s \equiv t$. The XOR-equivalence relation $=_{\mathsf{XOR}}$ on terms is defined as usual as the smallest congruence relation satisfying the following properties for all terms s, t, u.

$s \oplus (t \oplus u) =_{\mathsf{XOR}} (s \oplus t) \oplus u$ (associativity)	$s \oplus \mathsf{0} =_{\mathsf{XOR}} s$ (the unit axiom)
$s \oplus t =_{\mathsf{XOR}} t \oplus s$ (commutativity)	$s \oplus s =_{\mathsf{XOR}} \mathsf{0}$ (nil-potence)

Given a finite set of predicates, each with a fixed arity, *atoms* are of the form $P(t_1, \ldots, t_n)$ where t_i are terms and P is a predicate. *Literals* are either *positive literals* $+A$ (or simply A) or *negative literals* $-A$, where A is an atom. The relation $=_{\mathsf{XOR}}$ is extended to atoms and literals as usual. We also write $M \in_{\mathsf{XOR}} S$ to mean that $M =_{\mathsf{XOR}} N$ for some N with $N \in S$. Similarly we define $\subseteq_{\mathsf{XOR}}$. $S_1 =_{\mathsf{XOR}} S_2$ means that $S_1 \subseteq_{\mathsf{XOR}} S_2$ and $S_2 \subseteq_{\mathsf{XOR}} S_1$. *Clauses* are finite disjunction of literals, also thought of as the set of those literals. *Horn clauses* are those in which at most one positive literal occurs which is then called the *head*. The remaining literals in the Horn clause compose the *body* of the clause. *Definite clauses* are those with exactly

one positive literal, and *negative clauses* are those with no positive literal. The clauses $A \vee -A_1 \vee \ldots \vee -A_n$ and $-A_1 \vee \ldots \vee -A_n$ are also written as $A \Leftarrow A_1 \wedge \ldots \wedge A_n$ and $\Leftarrow A_1 \wedge \ldots \wedge A_n$ respectively. *Substitutions* σ are functions from a finite set dom_σ of variables, called the domain of σ, to terms. $M\sigma$ denotes the application, defined as usual, of substitution σ to M, where M is a term, atom, literal, clause or a set of such objects. This is well-defined if the domain of σ contains the set of variables occurring in M, denoted $\mathsf{fv}(M)$. Terms, atoms, literals and clauses are *ground* if they do not contain variables, and substitutions are ground if they map all variables in their domain to ground terms.

An *interpretation* H is a set of ground atoms such that if $P(s_1, \ldots, s_n) \in H$ and $s_i =_{\mathsf{XOR}} t_i$ for $1 \leq i \leq n$ then $P(t_1, \ldots, t_n) \in H$. H *satisfies* a set S of clauses if for every clause $C \in S$ and every ground substitution σ, either some $A \in H$ with $+A$ occurring in $C\sigma$, or some $A \notin H$ with $-A$ occurring in $C\sigma$. H is then called a *model* of S. S is *satisfiable* if it has a model. A satisfiable set of Horn clauses always has a least (w.r.t. the subset relation) model, and a set of definite clauses is always satisfiable. If some $P(t)$ is in the least model then we also say that $P(t)$ *holds*, or P *accepts* t. The latter terminology is based on the view of P as an automaton-state (see e.g. [4], Chapter 7). The least model of a set S of definite clauses is in fact the smallest interpretation H such that if $A \Leftarrow \bigwedge_{i=1}^n A_i \in S$, if σ is a ground substitution, and if $A_i\sigma \in H$ for $1 \leq i \leq n$ then $A\sigma \in H$. Hence corresponding to every element A of the least model we have a finite tree-shaped justification. Each node in the tree represents a justification step where some clause and substitution are used to obtain an atom, based on the atoms at the children nodes. The root node produces the atom A. This tree will be called a *derivation* of A.

It is also possible to view Horn clauses as definite clauses. To do this, we choose a fresh nullary predicate $\perp$. Then a set S of Horn clauses is satisfiable iff $\perp$ is not in the least model of the set S' of definite clauses, where S' consists of the definite clauses of S, together with the clauses $\perp \Leftarrow \bigwedge_{i=1}^n A_i$ for every negative clause $\Leftarrow \bigwedge_{i=1}^n A_i$ of S.

A term, atom, literal or a clause M is called *one-variable* if it contains at most one-variable. In particular, there is no restriction on the number of times this variable occurs in M. A clause C is called *flat* if it is of the form

$$\bigvee_{i=1}^m \pm_i P_i(f_i(x_i^1, \ldots, x_i^{n_i})) \vee \bigvee_{j=1}^n \pm'_j Q_j(x_j)$$

where $m, m \geq 0$, and for $1 \leq i \leq m$ we have $\{x_i^1, \ldots, x_i^{n_i}\} = \mathsf{fv}(C)$ and $f_i \neq \oplus$. Here $\pm, \pm_i$ are elements of $\{+, -\}$. We define the class $\mathcal{C}_1^\oplus$ to consist of sets of the form

$$S \cup \{\mathsf{I}(\mathsf{x}_1 \oplus \mathsf{x}_2) \Leftarrow \mathsf{I}(\mathsf{x}_1) \wedge \mathsf{I}(\mathsf{x}_2)\}$$

where S is a set of one-variable Horn clauses and flat Horn clauses. $\mathcal{C}_1^\oplus$ is the class that we are interested in in this paper. Note that in the last clause the same predicate appears three times, and we have only one such clause in the set. This clause will be called the *addition clause* of the set, and I will be called the *addition predicate*. The XOR operator is not allowed in flat clauses. There is however no restriction on the occurrences of the XOR operator in one-variable clauses. Finally note that flat clauses and addition clauses contain only unary predicates, but no such restriction has been put on one-variable clauses. However encoding atoms $P(t_1, \ldots, t_n)$ as atoms $P'(f_n(t_1, \ldots, t_n))$ for fresh P' and f_n ensures that restricting one-variable clauses to contain only unary predicates

causes no loss of generality. Hence in this paper we will consider only unary predicates and nullary predicates (the latter will be introduced later for technical purposes).

2.1 Protocols

Terms represent messages exchanged in protocols. We assume that Γ also contains binary symbols $\langle _, _ \rangle$, $\{_\}_$, $[_]_$ representing pairing, and symmetric encryption and asymmetric encryption of messages respectively. Symmetric encryption means that the same key (which could be an arbitrary term) is used for encryption and decryption. For asymmetric encryption we consider finitely many constants K representing encryption keys, and we correspondingly have constants K^{-1} representing decryption keys. A *protocol rule* is of the form $r_1, \ldots, r_n \longrightarrow s$ where s and all r_i are messages, possibly non-ground. This rule denotes sending out of message s after receipt of messages $r_1, \ldots, r_n$. A *protocol* is a finite set of protocol rules. The assumption of *single blind copying* then says that each protocol rule contains at most one variable (which may occur anywhere any number of times in that rule). Any protocol rule can be executed any number of times. The adversary has full control over the network: all messages received by agents are actually sent by the adversary and all messages sent by agents are actually received by the adversary. The adversary can obtain new messages from messages he knows, e.g. by performing encryption and decryption. Accordingly we choose a unary predicate I to model messages known to the adversary. The protocol rule $r_1, \ldots, r_n \longrightarrow s$ is translated to the clause $\mathsf{I}(s) \Leftarrow \mathsf{I}(r_1) \wedge \ldots \wedge \mathsf{I}(r_n)$. Under the assumption of single blind copying it is clear that this is a one-variable clause. We need further clauses to express adversary capabilities. The clauses

$$\mathsf{I}(\{\mathsf{x}_1\}_{\mathsf{x}_2}) \Leftarrow \mathsf{I}(\mathsf{x}_1) \wedge \mathsf{I}(\mathsf{x}_2) \qquad \mathsf{I}(\mathsf{x}_1) \Leftarrow \mathsf{I}(\{\mathsf{x}_1\}_{\mathsf{x}_2}) \wedge \mathsf{I}(\mathsf{x}_2)$$
$$\mathsf{I}([\mathsf{x}_1]_{\mathsf{x}_2}) \Leftarrow \mathsf{I}(\mathsf{x}_1) \wedge \mathsf{I}(\mathsf{x}_2) \qquad \mathsf{I}(\mathsf{x}_1) \Leftarrow \mathsf{I}([\mathsf{x}_1]_{\mathsf{x}_2}) \wedge \mathsf{I}_1(\mathsf{x}_2) \qquad \mathsf{I}_1(K) \Leftarrow \mathsf{I}(K^{-1})$$

express the encryption and decryption abilities of the adversary. The last clause is added for all (finitely many) keys K used for asymmetric encryption. We have similar clauses for his pairing and unpairing abilities. All these are clearly flat clauses. The adversary's knowledge of other data c like agent's names, public keys, etc are expressed by clauses $\mathsf{I}(c)$. In the presence of an XOR operator, we also have the clause

$$\mathsf{I}(\mathsf{x}_1 \oplus \mathsf{x}_2) \Leftarrow \mathsf{I}(\mathsf{x}_1) \wedge \mathsf{I}(\mathsf{x}_2)$$

Clearly we need only one such clause and the same predicate occurs three times in the clause, as required by the class $\mathcal{C}_1^{\oplus}$. A message M is *known* to the intruder if $\mathsf{I}(M)$ is in the least model of the set of clauses. A message is *secret* if it is not known to the intruder. To check secrecy of M, we add clause $-\mathsf{I}(M)$ and check satisfiability of the resulting clause set. Our protocol model is similar to that of [1] and of several other works. It is suitable for Horn-clause based analysis, and also allows several possible abstractions, all *safe* in the sense that no security flaws are missed by the analysis.

Example 1. The following is a modified version of the example protocol of [5], written in informal notation ("$X \rightarrow Y : M$" represents sending of message M from agent X to agent Y). S_{AB} is some confidential data which A wants to send to B, K_B is the public key of B, and N^1 and N^2 are fresh nonces chosen by A and B respectively. We are interested in the secrecy of S_{AB}. To formalize the

$$\begin{array}{l} A \rightarrow B : \{N^1\}_{K_B} \\ B \rightarrow A : N^1 \oplus N^2 \\ A \rightarrow B : N^2 \oplus S \end{array}$$

protocol in our notation, we follow [6] to choose two honest agents h_1 and h_2 and a dishonest agent d. For each pair $u \neq v$ of agents we create protocol rules

$$\longrightarrow \{n^1_{uv}\}_{K_v} \qquad \{\mathsf{x}_1\}_{K_v} \longrightarrow \mathsf{x}_1 \oplus n^2_{uv} \qquad \mathsf{x}_1 \longrightarrow \mathsf{x}_1 \oplus n^1_{uv} \oplus S_{uv}$$

This is based on the (safe) abstraction of using a single constant n^1_{uv} in place of infinitely many different nonces chosen by u in those sessions in which u starts a session with v, and similarly for n^2_{uv}. However in this formalization $S_{h_1h_2}$ is not secret. Since we do a safe abstraction, this merely indicates the possibility of a security flaw. We now consider a milder abstraction and use the following rules in place of the above.

$$\longrightarrow \{n^1_{uv}\}_{K_v} \qquad \{\mathsf{x}_1\}_{K_v} \longrightarrow \mathsf{x}_1 \oplus n^2_{uv}(\mathsf{x}_1) \qquad \mathsf{x}_1 \longrightarrow \mathsf{x}_1 \oplus n^1_{uv} \oplus S_{uv}$$

Now the nonce chosen by v is $n^2_{uv}(x)$, which is a function of the received nonce x (this is inspired from [1]). In this formalization, $S_{h_1h_2}$ is a secret, meaning that the flaw in the former formalization was introduced by the abstraction. Both formalizations are single blind copying protocols, and hence can be dealt with using our techniques. □

2.2 Related classes

We now consider some related classes of protocols, whose study is also proposed in [5,8]. Firstly we consider the cases where some other equational theory instead of XOR is used. We consider the theories AC, ACU and the theory AG of Abelian groups. The ACU theory is obtained by removing the nil-potence axiom from the XOR theory, i.e. we have the associativity axiom $s \oplus (t \oplus u) =_{\mathsf{ACU}} (s \oplus t) \oplus u$, the commutativity axiom $s \oplus t =_{\mathsf{ACU}} t \oplus s$ and unit axiom $s \oplus 0 =_{\mathsf{ACU}} s$. AC is obtained by removing the unit axiom from ACU. AG is obtained from ACU by adding an additional axiom $s \oplus -s =_{\mathsf{AG}} 0$ where $-$ is a unary symbol. We will see that if any of these theories is considered in place of the XOR theory, then we get undecidability, for the protocol analysis problem as well as for the satisfiability problem for the clauses. For the sake of this discussion, note that interpretations and satisfiability, as defined above w.r.t. the $=_{\mathsf{XOR}}$ relation should be redefined for the new relations $=_{\mathsf{ACU}}$ and others. In this and in later sections, we sometimes treat s and t as the same object when $s =_{\mathsf{AC}} t$. We write $\Sigma^n_{i=1} s_i$ to mean $s_1 \oplus \ldots \oplus s_n$. For $n = 0$ it denotes 0. We write nt to mean $\Sigma^n_{i=1} t$.

To start with we show that one-variable clauses in presence of the ACU theory become undecidable. We do this by encoding *two-counter automata*, in the style of [17]. We can conveniently define a two-counter automaton using Horn clauses. *Transitions* are clauses of the following form for arbitrary predicates P, Q.

$$\begin{aligned}
P(\mathsf{f}(\mathsf{x}_1), \mathsf{x}_2) &\Leftarrow Q(\mathsf{x}_1, \mathsf{x}_2) && \text{(increment first counter)}\\
P(\mathsf{x}_1, \mathsf{g}(\mathsf{x}_2)) &\Leftarrow Q(\mathsf{x}_1, \mathsf{x}_2) && \text{(increment second counter)}\\
P(\mathsf{x}_1, \mathsf{x}_2) &\Leftarrow Q(\mathsf{f}(\mathsf{x}_1), \mathsf{x}_2) && \text{(decrement first counter)}\\
P(\mathsf{x}_1, \mathsf{x}_2) &\Leftarrow Q(\mathsf{x}_1, \mathsf{g}(\mathsf{x}_2)) && \text{(decrement second counter)}\\
P(0, \mathsf{x}_2) &\Leftarrow Q(0, \mathsf{x}_2) && \text{(zero test on first counter)}\\
P(\mathsf{x}_1, 0) &\Leftarrow Q(\mathsf{x}_1, 0) && \text{(zero test on second counter)}
\end{aligned}$$

A *configuration* is of the form $P(\mathsf{f}^m(0), \mathsf{g}^n(0))$ with $m, n \geq 0$. A *two-counter automaton* is a finite set of transitions together with a configuration (representing the *initial* configuration of the machine). The set of *reachable* configurations of a two-counter automaton is defined to be the least model of its set of transitions together with the clause $P(s, t)$ where $P(s, t)$ is the initial configuration.

Checking whether a given configuration is reachable in a given two-counter automaton is undecidable. We now reduce it to satisfiability of one-variable clause sets modulo ACU. We encode configurations $P(\mathsf{f}^m(0), \mathsf{g}^n(0))$ as $P(m\mathsf{a} \oplus n\mathsf{b})$ where a and b are nullary symbols. The first, third and fifth clause above are then translated as the following clauses, and the second, fourth and sixth clauses are translated similarly:
$P(\mathsf{x}_1 \oplus \mathsf{a}) \Leftarrow Q(\mathsf{x}_1) \quad P(\mathsf{x}_1) \Leftarrow Q(\mathsf{x}_1 \oplus \mathsf{a}) \quad P(\mathsf{x}_1) \Leftarrow Q(\mathsf{x}_1) \wedge zero_1(\mathsf{x}_1)$
where $zero_1$ is a fresh auxiliary predicate corresponding to the first counter being zero. Accordingly we add clauses $zero_1(0)$ and $zero_1(\mathsf{x}_1 \oplus \mathsf{b}) \Leftarrow zero_1(\mathsf{x}_1)$.

Corresponding to the initial configuration $P(\mathsf{f}^m(0), \mathsf{g}^n(0))$ we add clause $P(m\mathsf{a} \oplus n\mathsf{b})$, and for the configuration $P(\mathsf{f}^p(0), \mathsf{g}^q(0))$ whose reachability we want to check, we add the clause $-P(p\mathsf{a} \oplus q\mathsf{b})$. Then unsatisfiability of this clause set is equivalent to reachability of the given configuration. A similar encoding works for AC and AG. In case of AG we have to explicitly prevent counters becoming negative during transitions. Hence the decrement of the first counter is translated as $P(\mathsf{x}_1) \Leftarrow Q(\mathsf{x}_1 \oplus \mathsf{a}) \wedge valid(\mathsf{x}_1)$ where the fresh auxiliary predicate $valid$ corresponds to valid configurations. Hence we add clauses $valid(0)$, $valid(\mathsf{x}_1 \oplus \mathsf{a}) \Leftarrow valid(\mathsf{x}_1)$ and $valid(\mathsf{x}_1 \oplus \mathsf{b}) \Leftarrow valid(\mathsf{x}_1)$.

Theorem 1. *Let $\mathcal{E}$ be the theory AC, ACU or AG. Then satisfiability of sets of one-variable clauses modulo $\mathcal{E}$ is undecidable. In particular the satisfiability for the class $\mathcal{C}_1^{\oplus}$ modulo $\mathcal{E}$ is undecidable.*

We also note that a similar undecidability results can be obtained for other theories e.g. those modeling associativity and commutativity of multiplication of exponents in the modular exponentiation operation.

We note that the above undecidability result can also be proved for the protocols, by reducing satisfiability of one-variable clauses to secrecy of protocols. The idea is same as the one used in [13,16] for proving lower bounds. Consider given some set of one-variable clauses. We choose a public data c, i.e. which is known to the adversary, and a symmetric key K not known to the adversary. Encode atoms $P(t)$ as messages $\{\langle P, t\rangle\}_K$, by treating P as some data. Corresponding to every clause $P(t) \Leftarrow P_1(t_1) \wedge \ldots P_n(t_n)$ for $n \geq 1$ we create the rule $\{\langle P_1, t_1\rangle\}_K, \ldots, \{\langle P_n, t_n\rangle\}_K \longrightarrow \{\langle P, t\rangle\}_K$. Corresponding to every clause $\Leftarrow P_1(t_1) \wedge \ldots \wedge P_n(t_n)$ for $n \geq 1$ we create the rule $\{\langle P_1, t_1\rangle\}_K, \ldots, \{\langle P_n, t_n\rangle\}_K \longrightarrow \{c\}_K$. The intuition is to let the adversary know messages $\{P, M\}_K$ exactly when $P(M)$ is in the least model, and to let him know $\{c\}_K$ exactly when the clause set is unsatisfiable. This works because the adversary cannot decrypt messages encrypted with K. He also cannot encrypt messages with K. He can only forward messages encrypted with K. Then secrecy of $\{c\}_K$ is equivalent to satisfiability of the clause set.

Theorem 2. *Let $\mathcal{E}$ be the theory AC, ACU or AG. Secrecy of cryptographic protocols with single blind copying is undecidable in presence of the theory $\mathcal{E}$.*

Returning to the XOR theory, we now consider relaxing the condition that only one addition clause $\mathsf{I}(x \oplus y) \Leftarrow \mathsf{I}(x) \wedge \mathsf{I}(y)$ is allowed. Let the class $\mathcal{C}_2^{\oplus}$ consist of sets of one-variable Horn clauses, flat Horn clauses and clauses of the form $P(x \oplus y) \Leftarrow Q(x) \wedge R(y)$. Note that P, Q, R need not be equal, and any number of such clauses are allowed in a set. This is another class whose decidability is conjectured in [5,8]. We

again show how to encode two counter automata. Configurations $P(\mathsf{f}^m(0), \mathsf{g}^n(0))$ are encoded as $P(\mathsf{f}^m(\mathsf{a}) \oplus \mathsf{g}^n(\mathsf{b}))$. The increment transition $P(\mathsf{f}(\mathsf{x}_1), \mathsf{x}_2) \Leftarrow Q(\mathsf{x}_1, \mathsf{x}_2)$ is translated to following clauses which use fresh auxiliary predicates and symbol f'.

$is_1(\mathsf{a})$ $\quad$ $is_1(\mathsf{f}(\mathsf{x}_1)) \Leftarrow is_1(\mathsf{x}_1)$	$is_2(\mathsf{b})$ $\quad$ $is_2(\mathsf{g}(\mathsf{x}_1)) \Leftarrow is_2(\mathsf{x}_1)$
$Qinc_1(\mathsf{x}_1 \oplus \mathsf{x}_2) \Leftarrow Q(\mathsf{x}_1) \wedge inc_1(\mathsf{x}_2)$	$inc_1(\mathsf{x}_1 \oplus \mathsf{f}'(\mathsf{x}_1)) \Leftarrow is_1(\mathsf{x}_1)$
$Qcheck_1(\mathsf{x}_1) \Leftarrow Qinc_1(\mathsf{x}_1) \wedge check_1(\mathsf{x}_1)$	
$check_1(\mathsf{x}_1 \oplus \mathsf{x}_2) \Leftarrow isf'(\mathsf{x}_1) \wedge is_2(\mathsf{x}_2)$	$isf'(\mathsf{f}'(\mathsf{x}_1))$
$Pfin(\mathsf{x}_1 \oplus \mathsf{x}_2) \Leftarrow Qcheck_1(\mathsf{x}_1) \wedge fin_1(\mathsf{x}_2)$	$fin_1(\mathsf{f}'(\mathsf{x}_1) \oplus \mathsf{f}(\mathsf{x}_1))$
$P(\mathsf{x}_1) \Leftarrow Pfin(\mathsf{x}_1) \wedge check(\mathsf{x}_1)$	$check(\mathsf{x}_1 \oplus \mathsf{x}_2) \Leftarrow is_1(\mathsf{x}_1) \wedge is_2(\mathsf{x}_2)$

If Q accepts $\mathsf{f}^m(\mathsf{a}) \oplus \mathsf{g}^n(\mathsf{b})$ then $Qinc_1$ should accept $\mathsf{f}'(\mathsf{f}^m(\mathsf{a})) \oplus \mathsf{g}^n(\mathsf{b})$ since inc_1 accepts $\mathsf{f}^m(\mathsf{a}) \oplus \mathsf{f}'(\mathsf{f}^m(\mathsf{a}))$. However $Qinc_1$ also accepts other unnecessary terms which are not of the form $\mathsf{f}'(x) \oplus \mathsf{g}^n(\mathsf{b})$ (e.g. $\mathsf{f}^m(\mathsf{a}) \oplus \mathsf{g}^n(\mathsf{b}) \oplus \mathsf{f}^{m+1}(\mathsf{a}) \oplus \mathsf{f}'(\mathsf{f}^{m+1}(\mathsf{a}))$). We filter these out and $\mathsf{f}'(\mathsf{f}^m(\mathsf{a})) \oplus \mathsf{g}^n(\mathsf{b})$ is accepted at $Qcheck_1$. We then finish the increment operation by accepting the term $\mathsf{f}(\mathsf{f}^m(\mathsf{a})) \oplus \mathsf{g}^n(\mathsf{b})$ at $Pfin$. $Pfin$ again accepts some bad terms not representing valid configurations. These are filtered out and $\mathsf{f}(\mathsf{f}^m(\mathsf{a})) \oplus \mathsf{g}^n(\mathsf{b})$ is accepted at P. Decrement transitions are translated using similar ideas. Zero test $P(0, \mathsf{x}_2) \Leftarrow Q(0, \mathsf{x}_2)$ is translated to clause $P(\mathsf{x}_1) \Leftarrow Q(\mathsf{x}_1) \wedge is_2(\mathsf{x}_1)$.

Theorem 3. *Satisfiability for the class $\mathcal{C}_2^{\oplus}$ modulo XOR is undecidable.*

To avoid undecidability, we may consider restricting the above clauses by enforcing that $P = Q = R$ in the above clause. In other words, we allow arbitrarily many clauses of the form $P(\mathsf{x}_1 \oplus \mathsf{x}_2) \Leftarrow P(\mathsf{x}_1) \wedge P(\mathsf{x}_2)$. This is the class $\mathcal{C}^{\oplus}$ considered in [5,8]. To be precise, $\mathcal{C}^{\oplus}$ allows the flat and one-variable clauses to be non-Horn. Compared to $\mathcal{C}^{\oplus}$, our class $\mathcal{C}_1^{\oplus}$ differs in two respects:

- non-Horn flat and one-variable clauses are not allowed in $\mathcal{C}_1^{\oplus}$.
- only one clause of the form $P(\mathsf{x}_1 \oplus \mathsf{x}_2) \Leftarrow P(\mathsf{x}_1) \wedge P(\mathsf{x}_2)$ is allowed in $\mathcal{C}_1^{\oplus}$.

An algorithm for the class $\mathcal{C}^{\oplus}$ is proposed in [8,5] where a non-elementary time complexity upper bound is claimed for the satisfiability problem (and consequently for the protocol verification problem). The proof seems to have some problems, as confirmed to us by V. Cortier [9]. Hence decidability of the class $\mathcal{C}^{\oplus}$ is open to the best of our knowledge. Instead we focus on the restricted class $\mathcal{C}_1^{\oplus}$ which suffices for the modeling of protocols. The decision procedure for $\mathcal{C}_1^{\oplus}$ presented in this paper means that decidability of the protocol verification problem for single blind copying protocols with XOR can indeed be shown, and in fact a triple exponential complexity upper bound can also be obtained. Our techniques do not seem to apply for the class $\mathcal{C}^{\oplus}$ even if only Horn clauses are considered. A key idea we use for showing the 3-EXPTIME upper bound is decomposition of one-variable terms into simpler terms. We introduced this idea in [16] for the non-XOR case, and generalize it in this paper to the XOR case.

Other related work. Several other work deal with classes of protocols and of Horn clauses in the presence of XOR [11,14,17,18,19]. These are incomparable to the class of single blind copying protocols with XOR, and to the class $\mathcal{C}_1^{\oplus}$. In [17,18,19], decidability and complexity of classes of Horn clauses modulo various algebraic properties including XOR are presented. In the absence of equational theories, these classes correspond to one-way and two-way alternating tree automata, in the style of [4] (see Chapter

7). Compared to $\mathcal{C}_1^{\oplus}$, the two-way automata clauses contain only linear terms, but allow clauses of the form $P_1(x \oplus y) \Leftarrow P_2(x) \wedge P_3(y)$ without any restriction. A decision procedure for XOR-based key management schemes is presented in [11]. Compared to the class studied here, they forbid nested encryption in protocol steps, but allow more than one variables in protocol steps. In [14], again following the approach based on Horn clauses, the verification problem for a general class of protocols with XOR is reduced to that of protocols without XOR. The syntax of protocols is restricted in such a way that the XOR operator is never applied to two non-ground terms. Hence although the protocol steps of [14] may contain more than one variables, terms like $f(x) \oplus g(x)$ are forbidden but are allowed in our one-variable clauses. For example, the two formalizations of the protocol in Example 1 lie in our class but the second one (which turned out to be necessary for showing secrecy) cannot be dealt with by [14]. Hence the reduction of [14], combined with the decidability result on single blind copying protocols without XOR, still does not yield the decidability results which we prove here. The goal of [14] is not to obtain new decidability results, but to use tools like ProVerif [1] which are efficient in practice in the non-XOR case.

3 Results on Unification

If M is a term, literal, clause, or sets of terms or literals, and if $\mathsf{fv}(M) \subseteq \{x\}$ then we also write $M[x]$ to emphasize this. $M[t]$ then denotes $M\sigma$ where $\sigma = \{x \mapsto t\}$. This is extended to sets as usual. Hence for sets X and Y of one-variable terms we define $X[Y] = \{s[t] \mid s \in X, t \in Y\}$. We also write $M \circ N$ to mean $M[N]$. Define $X^n = \{s_1 \circ \ldots \circ s_n \mid \text{ each } s_i \in X\}$, $X^0 = \{x\}$, $X^* = \bigcup_{n \geq 0} X^n$, $X \oplus Y = \{s \oplus t \mid s \in X, t \in Y\}$, $nX = \{s_1 \oplus \ldots \oplus s_n \mid \text{ each } s_i \in X\}$, and $\Sigma X = \bigcup_{n \geq 0} nX$. $\{t\} \oplus M$ will also be written $t \oplus M$. *Trivial* terms are those of the form $x \oplus g$ for some variable x and some ground term g. Trivial literals are those of the form $\pm P(x \oplus g)$ and trivial clauses are those containing only trivial literals. Terms are viewed as trees, and positions or nodes in the trees are sequences of strictly positive integers, as usual. The root node is denoted by the empty sequence λ. The term occurring at a position π in a term t is denoted as $t|_\pi$. The symbols $\oplus$ and 0 are *equational*. Other symbols are *free*, and are denoted using $f, g, \ldots$. If M is a symbol (resp. a set of symbols) then an M-term is a term containing M (resp. a symbol from M) at the root, and an M-position in a term is a position labeled with M (resp. a symbol from M). The notions of *free terms*, *free positions*, *equational terms* and *equational positions* are defined accordingly. Given a term t, if π is a position in t then a $\oplus$*-child* of π is a position δ such that δ is not labeled with $\oplus$, $\pi \leq \delta$, and for every $\pi \leq \delta' < \delta$, δ' is labeled with $\oplus$. In this case, π is the $\oplus$*-parent* of δ. If δ_1 and δ_2 are mutually distinct $\oplus$-children of π then δ_1 and δ_2 are $\oplus$*-siblings*. A *summand* of a term t is a term at a $\oplus$-child of λ in t. The *subterm* relation on terms is defined on terms modulo $=_{\mathsf{AC}}$.

- t is a subterm of t.
- If s is a subterm of some t_i then s is a subterm of $f(t_1, \ldots, t_n)$ where f is free.
- If s is a subterm of t and if t is not a $\oplus$-term then s is a subterm of $t \oplus u$.

Hence $a \oplus b$ is not a subterm of $a \oplus b \oplus c$. We say that s is a strict subterm of t if $t =_{\mathsf{ACU}} u \oplus f(s_1, \ldots, s_n)$ for some free f and s is a subterm of some s_i. A term is

ACU-*normal* if it has no subterm of the form $u \oplus 0$. A term t is XOR-*normal* (or simply normal) if it has no two $\oplus$-sibling positions π_1 and π_2 with $t|_{\pi_1} = t|_{\pi_2}$. We will denote by $t\Downarrow$ the ACU-normal form of a term t and by $t\downarrow$ the XOR-normal form of a term t, both unique upto $=_{\mathsf{AC}}$.

Unifiers and *most general unifiers (mgus)* are defined w.r.t. the XOR theory as usual. Note that mgus need not be unique, unlike in the non-XOR case. A substitution σ is *trivial* if $x\sigma =_{\mathsf{XOR}} y \oplus g$ for some ground term g. A trivial instance of a term, clause, or substitution M is of the form $M\sigma$ where σ is trivial. Note that if substitution σ_1 is a trivial instance of σ_2 then σ_2 is also a trivial instance if σ_1. When defining mgus, it is unnecessary to include all trivial instances. We will arbitrarily choose some of them to keep and leave others. Unifying two distinct one-variable terms with the same variable produces ground unifiers of a simple form.

Lemma 1. *Let s and t be two normal one-variable terms such that $s[x] \neq t[x]$. For every unifier σ of $s[x]$ and $t[x]$, we have $x\sigma \in_{\mathsf{XOR}} \Sigma G$ where G is the set of ground subterms of s and t.*

Contrary to what is conjectured in [8], there can be more than one unifier in the case above. For example, the terms $f(x) \oplus f(x \oplus a)$ and $f(a) \oplus f(0)$ have $\{x \mapsto a\}$ as well as $\{x \mapsto 0\}$ as unifiers. In the above result, the summands are in fact strict subterms of s and t, under a simple condition. This is stated below.

Lemma 2. *Let s and t be two normal one-variable terms such that $s[x] \neq t[x]$ and $(s[x] \oplus t[x])\downarrow$ is non-ground and non-trivial. Then any unifier σ of $s[x]$ and $t[x]$ satisfies $x\sigma \in_{\mathsf{XOR}} \Sigma G$ where G is the set of free ground strict subterms of s and t.*

For a term t, $\|t\|$ denotes the maximum depth at which variables appear in t, and is defined as $\|x\| = 1$ for all variables x, $\|t\| = 0$ when t is ground, $\|t_1 \oplus t_2\| = max\{\|t_1\|, \|t_2\|\}$, and $\|f(t_1, \ldots, t_n)\| = 1 + max_{i=1}^{n}\|t_i\|$ when f is free and at least one t_i is non-ground. The depth $\|L\|$ of a literal L is defined as $\|P\| = 0$ if P is a nullary predicate, and $\|P(t)\| = \|t\|$. The depth of a clause is defined as $\|\bigvee_{i=1}^{n} L_i\| = \max_{i=1}^{n} \|L_i\|$, and is 0 for the empty clause. As stated by Lemma 1, two one-variable terms over the same variable cannot be made equal except by a ground substitution. Consequently, the $\|.\|$ operator has the following nice property w.r.t. one-variable terms.

Lemma 3. $\|s[t[x]]\downarrow\| = \|s[x]\| + \|t[x]\| - 1$ *for normal non-ground terms s and t.*

We now consider unification of one-variable terms not sharing variables.

Lemma 4. *Let s and t be normal one-variable terms with $x \neq y$ and $s[x] \neq t[y]$, and σ a most general unifier of $s[x]$ and $t[y]$. Let G be the set of ground subterms of s and t, and Ng the set of non-ground subterms of s and t. Then one of the following is true.*

- $x\sigma \in_{\mathsf{XOR}} Ng[y] \oplus \Sigma G$ *and* $y\sigma = y$,
- $y\sigma \in_{\mathsf{XOR}} Ng[x] \oplus \Sigma G$ *and* $x\sigma = x$,
- $x\sigma, y\sigma \in_{\mathsf{XOR}} (\Sigma Ng)[\Sigma G]$

It is interesting to note that $\{x \mapsto y\}$ need not be the only mgu of $s[x]$ and $t[y]$, even when $s[x] = t[x]$. For example $f(x) \oplus f(x \oplus a)$ and $f(y) \oplus f(y \oplus a)$ have $\{x \mapsto y\}$ as well as $\{x \mapsto y \oplus a\}$ as mgus.

We are now ready to define decomposition of one-variable terms. We did this in [16] for the non-XOR case, and generalize it now to the XOR case. We think of one-variable terms as strings like $f_1(x) \circ f_2(x) \circ f_3(x)$ and $f_1(x) \circ a$. To unify two strings, we match the two initial symbols, and then unify the remaining parts. A non-ground non-trivial one variable term $s[x]$ is *reducible* if there are one-variable terms s_1 and s_2, with $s_1\downarrow$ and $s_2\downarrow$ both non-ground and non-trivial, such that $s[x] =_{\mathsf{XOR}} s_1[x] \circ s_2[x]$. Otherwise it is called *irreducible*. In the analogy with strings, irreducible terms are like symbols of the strings. The term $x \oplus f(x)$ is irreducible. The term $t = h(x \oplus f(x)) \oplus x \oplus f(x)$ is reducible because $t = (h(x) \oplus x) \circ (x \oplus f(x))$. Viewing terms as composed of irreducible terms is critical for obtaining a 3-EXPTIME upper bound, since a linear bound on $\|.\|$ of terms generated only gives a non-elementary upper bound. The following result says that mgus of irreducible terms are trivial or ground.

Lemma 5. *Let $s[x]$ and $t[y]$ be non-ground irreducible normal terms. Let G be the set of ground subterms of s and t, and Ng the set of non-ground subterms of s and t. Then mgus σ of $s[x]$ and $t[y]$ are of the following form.*

- $x\sigma \in_{\mathsf{XOR}} y \oplus \Sigma G$ *and* $y\sigma = y$.
- $x\sigma, y\sigma \in_{\mathsf{XOR}} (\Sigma Ng)[\Sigma G]$.

Proof. We consider the three cases in the conclusion of Lemma 4. In the third case the proof is done. In the first case, if $x\sigma$ is non-trivial then we get $s[x\sigma] = t[y]$ which is a contradiction since t is irreducible. Similarly in the second case, $y\sigma = x \oplus g, x\sigma = x$ for some $g \in \Sigma G$. But then σ is an instance of $\{x \mapsto y \oplus g, y \mapsto y\}$ which is also a unifier and can hence be chosen as the unifier instead. □

Now for the rest of the paper, we fix a set S_0 of clauses with $\mathsf{S}_0 \in \mathcal{C}_1^{\oplus}$. We will show how to decide satisfiability of S_0. We assume all terms occurring in S_0 are normal. Let G be the set of normal forms of finite summations of ground subterms of terms occurring as arguments of predicates in S_0 and $\mathsf{G}_1 = \{f(s_1, \ldots, s_m) \mid$ some $g(t_1, \ldots, t_n) \in \mathsf{G}$ with $\{s_1, \ldots, s_m\} = \{t_1, \ldots, t_m\}$, f and g both free$\}$. Let Ngr be the set of non-ground non-trivial irreducible terms $t[x] \oplus g$ such that there is some non-ground term $s[x]$ such that $t[s]$ is a subterm of a term occurring as an argument of a predicate in S_0, and $g \in \mathsf{G}$. Note that if u is an irreducible term then $u \oplus g$ is irreducible for all ground g. Let Ngs be the set of non-ground subterms of terms in Ngr. Let $\mathsf{G}_2 = (\Sigma\mathsf{Ngs})[\mathsf{G}]$ and $\mathsf{Ngr}_1 = \{f(t_1, \ldots, t_n) \mid g(s_1, \ldots, s_m) \in \mathsf{Ngr}$ and $\{t_1, \ldots, t_n\} = \{s_1, \ldots, s_m\}\}$. Note that if $\{t_1, \ldots, t_n\} = \{s_1, \ldots, s_m\}$ and if $g(s_1, \ldots, s_m)$ is irreducible then $f(t_1, \ldots, t_n)$ is also irreducible, where f and g are free. The above sets are defined modulo renaming of variables.

Example 2. Let clause $P(f(h(x) \oplus a, h(x) \oplus g(h(x)))) \Leftarrow Q(x \oplus b \oplus f'(b, b, c))$ be in S_0. Then terms a, b, c, $f'(b, b, c)$, $b \oplus f'(b, b, c)$, $a \oplus f'(b, b, c)$, $a \oplus b \oplus c$,... are present in G. Terms $f'(b, b, c)$, $f'(c, b, c)$, $f(b, c)$,... are present in G_1. Ngr contains terms like $f(x \oplus a, x \oplus g(x))$, $x \oplus g(x)$, $h(x)$, $g(x)$, $g(x) \oplus f'(b, b, c)$. Ngr_1 contains

terms like $f(x \oplus a, x \oplus g(x))$, $f(x \oplus g(x), x \oplus a)$, $f'(x \oplus g(x), x \oplus g(x), x \oplus a)$. If the clause $R(f(x, h(h(x))))$ is in S_0 then $f(x, h(h(x))) \in \mathsf{Ngr}$, $h(h(x)) \in \mathsf{Ngs}$ but $h(h(x)) \notin \mathsf{Ngr}$. □

We note some useful properties of these sets.

- The terms in $\mathsf{Ngr} \cup \mathsf{Ngr}_1$ are irreducible.
- The sets G, G_1, G_2, Ngr and Ngr_1 are of exponential size.
- $\Sigma\mathsf{G} =_{\mathsf{XOR}} \mathsf{G}$, $\mathsf{Ngr} \oplus \mathsf{G} =_{\mathsf{XOR}} \mathsf{Ngr}$, and $\mathsf{Ngs} \subseteq_{\mathsf{XOR}} \mathsf{Ngr}^*$.
- Subterms of terms in G are again in G.
- Ground subterms of terms in $\mathsf{Ngr}, \mathsf{Ngr}^*, \mathsf{Ngr}_1, \mathsf{Ngr}_1[\mathsf{Ngr}^*]$ are in G.
- Strict subterms of terms in G_1 are in G

Let $\mathcal{K} = max_{C \in \mathsf{S}_0} \|C\|$, i.e. the maximal variable depth of the clauses in the input set. We introduce the following kinds of terms which will occur in clauses generated by the normalization algorithm of the next section. Recall that we view one-variable terms as sequences of irreducible terms. The first idea is to consider clauses having just these sequences. But since irreducible terms can also have ground unifiers, we also need to consider the appropriate ground instances of these terms. The irreducible terms in our case are those from Ngr. However interaction between flat and one-variable clauses also produces irreducible terms in Ngr_1. For example let $s[x] = f(x, g(x))$ and $t[x] = h(x)$ be both in Ngr. Given clauses $P(x) \Leftarrow Q(s[t[x]])$ and $Q(f(y, z)) \Leftarrow R(f'(y, y, z))$, the normalization algorithm of the next section will produce clause $P(x) \Leftarrow R(s'[t[x]])$ where $s'[x] = f'(x, x, g(x)) \in \mathsf{Ngr}_1$. Let $\mathcal{NG}$ be the set of terms having non-ground normal forms. We now define

$$\begin{array}{llll}
\mathsf{Ov} & = \mathsf{Ngr}_1[\mathsf{Ngr}^*] & \mathsf{Gov} & = \mathsf{Ov} \circ \mathsf{G}_2 \\
\mathsf{Ov}^\natural & = \{s \in \mathsf{Ov} \mid \|s\| \leq \mathcal{K}\} & \mathsf{Gov}^\natural & = \mathsf{Ov}^\natural \circ \mathsf{G}_2 \\
\mathcal{T}_1 & = \Sigma(\mathsf{Ov} \cup \{\mathsf{x}_1\} \cup \mathsf{G}) \cap \mathcal{NG} & \mathcal{T}_2 & = \Sigma(\mathsf{Gov} \cup \mathsf{G}_1) \\
\mathcal{T}_1^\natural & = \Sigma(\mathsf{Ov}^\natural \cup \{\mathsf{x}_1\} \cup \mathsf{G}) \cap \mathcal{NG} & \mathcal{T}_2^\natural & = \Sigma(\mathsf{Gov}^\natural \cup \mathsf{G}_1) \\
\multicolumn{2}{l}{\mathsf{F} = \text{set of terms of the form } f(x_1, \ldots, x_n) \text{ with free } f} & \mathcal{T}_3 & = \mathsf{F} \oplus \mathsf{G}
\end{array}$$

The terms with bounded depth have been defined above in order to show that we produce only those finitely many terms. Note that if σ unifies s and t then $s\sigma \oplus t\sigma =_{\mathsf{XOR}} 0$. In the following lemma we generalize this situation to n terms, i.e. when we have $\Sigma_{i=1}^n s_i\sigma =_{\mathsf{XOR}} 0$. This is an important technical lemma which helps us in dealing with the addition clause. Intuitively the lemma says that in such a situation, with some exceptions, we must have maximal (w.r.t. $\|.\|$) summands u and v of some s_i and $s_{i'}$ respectively, such that $u\sigma =_{\mathsf{XOR}} v\sigma$. The exceptions occur when some summands of some $s_i \in \mathcal{T}_1$ get canceled out using the nil-potence axiom, or when some term in F unifies with some free term.

Lemma 6. *Let $n \geq 1$ and $s_i \in \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$ be non-trivial normal terms for $1 \leq i \leq n$, mutually renamed apart, and σ a substitution such that $\sum_{i=1}^n s_i\sigma =_{\mathsf{XOR}} 0$. For each i, let $s_i = \sum_{j=1}^{k_i} s_i^j$ such that each s_i^j is non-equational. Let $s_i \neq 0$ for some i. Then one of the following is true, where x_i is the free variable of s_i whenever $s_i \in \mathcal{T}_1$.*

- *for some i, $s_i \in \mathcal{T}_1$ and $\mathsf{x}_i\sigma \in \mathsf{G}_2$.*
- *for some i, i', j, j', $i \neq i'$, we have $s_i, s_{i'} \in \mathcal{T}_1 \cup \mathcal{T}_2$, $s_i^j\sigma =_{\mathsf{XOR}} s_{i'}^{j'}\sigma$, $\|s_i^j\| = \|s_i\|$ and $\|s_{i'}^{j'}\| = \|s_{i'}\|$.*

- *for some $i \neq i'$, we have $s_i, s_{i'} \in \mathcal{T}_3$, $s_i^j, s_{i'}^{j'} \in \mathsf{F}$ and $s_i^j\sigma =_{\mathsf{XOR}} s_{i'}^{j'}\sigma$.*
- *$s_i = u \oplus g$ with $u \in \mathsf{F}$ and $g \in \mathsf{G}$ for some i such that $u\sigma \in \mathsf{G}_1 \cup \mathsf{Ngr}_1[\mathsf{G}_2]$ or $u\sigma$ is an instance of $u\sigma'$ for some σ' such that $u\sigma' \in \mathsf{Ngr}_1$.*

4 The Normalization Algorithm

We present a *normalization* algorithm for deciding satisfiability, i.e. the algorithm will produce an equivalent (w.r.t. least model) set of simple clauses called *normal clauses* on which decision problems become easy. Using the idea of Section 2, we consider only definite clauses. Satisfiability is equivalent to the nullary predicate $\bot$ not being present in the least model. This also means that our algorithm should deal with nullary predicates, and in fact we will need several more nullary predicates in our algorithm.

Blocks are conjunctions $\mathcal{B}[x]$ of the form $\bigwedge_{i=1}^n P_i(x \oplus g_i)$ with $n \geq 1$ and each $g_i \in \mathsf{G}$. We introduce a fresh nullary predicate $\widehat{\mathcal{B}}$ for each block $\mathcal{B}$. Here blocks are considered modulo renaming of variables and modulo $=_{\mathsf{XOR}}$ on terms. We further introduce fresh nullary predicates $\widehat{P(t)}$ for every predicate P and every $t \in \mathcal{T}_2$, again considered modulo $=_{\mathsf{XOR}}$. Let $\mathbb{P}_0$ be the set of these fresh nullary predicates as well as any nullary predicates originally present in S_0. We slightly generalize the syntax of flat and one-variable clauses to now allow nullary predicates. Accordingly we define $\mathsf{S}_{\mathsf{max}}$ to consist of definite clauses of the following form.

- the addition clause $\mathsf{I}(\mathsf{x}_1 \oplus \mathsf{x}_2) \Leftarrow \mathsf{I}(\mathsf{x}_1) \wedge \mathsf{I}(\mathsf{x}_2)$
- one-variable clauses $C \vee D$ where $D \subseteq \pm\mathbb{P}_0$ and $C \subseteq \pm\mathbb{P}(\mathcal{T}_1^\natural \cup \mathcal{T}_2^\natural)$,
- clauses $C \vee D$ where $D \subseteq \pm\mathbb{P}_0$ and C is a clause with at least one non-trivial literal, each literal of C is of the form $\pm P(x \oplus g)$ with $g \in \mathsf{G}$, or of the form $\pm P(t)$ with $t \in \mathcal{T}_3$ and $\mathsf{fv}(t) = \mathsf{fv}(C)$.

Above, $M(N)$ denotes a set of atoms for a set M of predicates and a set N of terms, and is defined as usual. The sets $-N, \pm N$ denote sets of literals for sets N of atoms, and are defined as usual. We have bounded the depth of one-variable terms in order to obtain our complexity upper bound. The third kind of clause above is called a *complex* clause. These are similar to flat clauses, but we disallow clauses like $P(\mathsf{x}_1) \Leftarrow Q(\mathsf{x}_2)$, and allow ground summands in the arguments of predicates. The following kinds of clauses are called *normal*.

- addition clause
- trivial clauses of the form $P(x \oplus g) \Leftarrow \bigwedge_{i=1}^n \mathsf{I}(x \oplus g_i)$
- clauses $A \Leftarrow \bigwedge_{i=1}^n A_i$ in which $\|A\| > \|A_i\|$ for all i.

Let $\mathsf{S}_1 = \mathsf{S}_0 \cup \{\widehat{P(t)} \Leftarrow P(t) \mid t \in \mathcal{T}_2\} \cup \{\widehat{\mathcal{B}} \Leftarrow \mathcal{B} \mid \mathcal{B} \text{ is a block}\}$. We can see that if H is the least model of S_0 then H' is the least model of S_1 where H' consists of the atoms in H together with the atoms $\widehat{P(t)}$ for all $P(t) \in H$ with $t \in \mathcal{T}_2$, as well as atoms $\widehat{\mathcal{B}}$ where $\mathcal{B} = \bigwedge_{i=1}^n P_i(x \oplus g_i)$ and there is some t such that $P_i(t \oplus g_i) \in H$ for all i. In particular $\bot \in H'$ iff $\bot \in H$.

The normalization algorithm uses a saturation procedure which adds more and more clauses, hopefully of simpler form, until all necessary normal clauses have been added.

The saturation steps are as follows. Note that all steps other than Step 1 just add new clauses, and they can be applied in some arbitrary order. However Step 1 replaces a clause by another clause, and it is also eager, i.e. no other step can be applied as long as this step is applicable.

1. Let $\mathcal{B}_2$ be a non-empty conjunction $\bigwedge_{i=1}^{n} P_i(x \oplus g_i)$, and let $A \Leftarrow \mathcal{B}_1$ be non-ground and not containing variable x. Clause $A \Leftarrow \mathcal{B}_1 \wedge \mathcal{B}_2$ is then *eagerly replaced* by $A \Leftarrow \mathcal{B}_1 \wedge \widehat{\mathcal{B}_2}$.
2. Let C be non-ground and t ground. From clause $C \vee -P(t)$ we obtain $C \vee -\widehat{P(t)}$.
3. Let P be a nullary predicate. From clauses $C \vee -P$ and P we obtain C.
4. Let $p \geq 1$. From the clause $\mathsf{I}(x \oplus g) \Leftarrow \bigwedge_{i=1}^{p} \mathsf{I}(x \oplus g_i)$ we obtain the clause $\mathsf{I}(g_1 \oplus g) \Leftarrow \bigwedge_{i=1}^{p} \mathsf{I}(\widehat{g_1 \oplus g_i})$.
5. Let $p \geq 1$ and A be a ground atom (possibly a nullary predicate). From clause $A \Leftarrow \bigwedge_{i=1}^{p} \mathsf{I}(x \oplus g_i)$ we obtain $A \Leftarrow \bigwedge_{i=1}^{p} \mathsf{I}(g_1 \oplus g_i)$.
6. From $C[x]$ we obtain $C[g]$ where C is a one-variable clause and $g \in \mathsf{G}_2$.
7. From complex clause C in which a term $f(x_1, \ldots, x_n)$ appears, we obtain $C\sigma$, provided $f(x_1, \ldots, x_n)\sigma \in \mathsf{Ngr}_1 \cup \mathsf{G}_1 \cup \mathsf{Ngr}_1[\mathsf{G}_2]$.
8. Let $C_1 = C_1' \vee P(s)$ and $C_2 = -P(t) \vee C_2'$ be complex or one-variable clauses, $\|P(s)\| = \|C_1\|$, $\|P(t)\| = \|C_2\|$, and σ be a mgu of s and t. From C_1 and C_2 we obtain $C_1'\sigma \vee C_2'\sigma$.
9. Let $u \downarrow\!=_{\mathsf{ACU}} u_1 \oplus u_2$, $s \downarrow\!=_{\mathsf{ACU}} s_1 \oplus s_2$, u_1 be free, $\|u_1\| \geq \|u_2\|$, s_1 be free, $\|s_1\| \geq \|s_2\|$, and σ be a mgu of u_1 and s_1. Let $\|\mathsf{I}(u)\| \geq \|C_1\|$ and $\|\mathsf{I}(s)\| \geq \|C_2\|$. From clauses $C_1 \vee -\mathsf{I}(u)$ and $\mathsf{I}(s) \vee C_2$ we obtain $C_1\sigma \vee -\mathsf{I}(u_2 \oplus s_2)\sigma \vee C_2\sigma$.
10. Let $u \downarrow\!=_{\mathsf{ACU}} u_1 \oplus u_2$, $s \downarrow\!=_{\mathsf{ACU}} s_1 \oplus s_2$, u_1 be free, $\|u_1\| \geq \|u_2\|$, s_1 be free, $\|s_1\| \geq \|s_2\|$, $\|u\| \geq \|C_1\|$ and $\|s\| \geq \|C_2\|$ and and σ be a mgu of u_1 and s_1. From clauses $\mathsf{I}(u) \vee C_1$ and $\mathsf{I}(s) \vee C_2$ we obtain $\mathsf{I}(u_2 \oplus s_2)\sigma \vee C_1\sigma \vee C_2\sigma$.

Step 5 explains why our techniques work for the class $\mathcal{C}_1^{\oplus}$ but not for the more general class $\mathcal{C}^{\oplus}$ of [5,8]. While clauses like $P(x \oplus a) \Leftarrow Q(x \oplus b) \wedge R(x \oplus c)$ are easy to deal with when $\{\mathsf{I}\} \neq \{P, Q, R\}$, clauses like $\mathsf{I}(x \oplus a) \Leftarrow \mathsf{I}(x \oplus b) \wedge \mathsf{I}(x \oplus c)$ are problematic. Fortunately, in presence of the addition clause, this is equivalent to the ground clause $\mathsf{I}(b \oplus a) \Leftarrow \mathsf{I}(0) \wedge \mathsf{I}(b \oplus c)$ which can then be replaced by $\mathsf{I}(b \oplus a) \Leftarrow \widehat{\mathsf{I}(0)} \wedge \widehat{\mathsf{I}(b \oplus c)}$. On the other hand, if we allowed clauses $\mathsf{I}_i(x \oplus y) \Leftarrow \mathsf{I}_i(x) \wedge \mathsf{I}_i(y)$ for $i = 1, 2, 3$ then the clause $\mathsf{I}_1(x \oplus a) \Leftarrow \mathsf{I}_2(x \oplus b) \wedge \mathsf{I}_3(x \oplus c)$ is difficult to deal with.

As is usual for saturation procedures, correctness of the rules is easy to show, since the new clauses are essentially logical implications of the old clauses. Steps 1 and 2 are correct because of the new clauses added to S_0 to obtain S_1. Steps 9 and 10 deal with addition clauses, and are essentially as used in [5,8]. But instead of resolution based reasoning as in [5,8], we analyze the derivation trees, and use Lemma 6 to reason about XOR of arbitrary number of terms. For example, given clauses $P(x) \Leftarrow \mathsf{I}(f(x) \oplus x)$ and $\mathsf{I}(f(y) \oplus g(y)) \Leftarrow R(y)$, Step 9 gives us the clause $P(x) \Leftarrow \mathsf{I}(x \oplus g(x)) \wedge R(x)$. Given clauses $\mathsf{I}(f(x) \oplus a) \Leftarrow P(x)$ and $\mathsf{I}(f(g(y)) \oplus h(y)) \Leftarrow Q(g(y))$, Step 10 gives us the clause $\mathsf{I}(h(y) \oplus a) \Leftarrow P(g(y)) \wedge Q(g(y))$. In both cases the new clause is a logical implication of the given clauses together with the addition clause $\mathsf{I}(x \oplus y) \Leftarrow \mathsf{I}(x) \wedge \mathsf{I}(y)$.

Lemma 7 (Correctness). *Let $\mathsf{S}_1 \subseteq S \subseteq \mathsf{S}_{\mathsf{max}}$. Let S' be obtained from S by applying one of Steps 1- 10. Then S and S' have the same least model.*

We next show that new clauses produced are of the right form. The essential idea is that we always choose maximal (w.r.t $\|.\|$) terms for unification, so that the clauses C produced have bounded $\|C\|$.

Lemma 8 (Preservation of syntax). *Let $S \subseteq \mathsf{S}_{\mathsf{max}}$. Let S' be obtained from S by applying one of Steps 2- 10 followed by applications of Step 1 as long as possible. Then $S' \subseteq \mathsf{S}_{\mathsf{max}}$.*

Our strategy now is to apply the saturation steps to add clauses of $\mathsf{S}_{\mathsf{max}}$ to S_1. This is continued till no new clauses can be added. The process must end because $\mathsf{S}_{\mathsf{max}}$ is clearly finite. Let $\mathsf{S}_{\mathsf{sat}}$ be the saturated set produced in this way. In $\mathsf{S}_{\mathsf{sat}}$, the non-normal clauses are redundant, i.e. have no impact on derivability of atoms.

Lemma 9 (Completeness). *Let the set $\mathsf{S}_{\mathsf{sat}}$ be saturated. Then any derivation Δ of an atom using clauses of $\mathsf{S}_{\mathsf{sat}}$ can be transformed to a derivation Δ_0 of the same atom involving only normal clauses of $\mathsf{S}_{\mathsf{sat}}$.*

A typical example case in usual completeness proofs is application of a normal clause $Q(f(x)) \Leftarrow R(x)$ followed by application of a non-normal clause $P(h(y)) \Leftarrow Q(f(g(y)))$. But the saturated set should then contain $P(h(y)) \Leftarrow R(g(y))$ which can be applied instead of the above two clauses to get a smaller derivation. Steps 9 and 10 are meant especially for the class $\mathcal{C}_1^{\oplus}$. They deal with the case where an atom $\mathsf{I}(\sum_{i=1}^{n} t_i)$ is derived from derivations of $\mathsf{I}(t_i), 1 \leq i \leq n$ using $n-1$ applications of the addition clause. Lemma 6 helps us to deal with the unifications involved in this case.

Recall that $\bot$ is in the least model of S_0 iff it is in the least model of S_1. And the latter holds iff $\bot$ is in the least model of $\mathsf{S}_{\mathsf{sat}}$. But in the latter case, there must be a normal derivation of $\bot$ which means that $\bot \in \mathsf{S}_{\mathsf{sat}}$. Hence we just need to check whether $\bot \in \mathsf{S}_{\mathsf{sat}}$. To analyze the time complexity of our algorithm we see that the sets $\mathsf{Ov}^{\natural}$, $\mathsf{Gov}^{\natural}$, G, G_1, G_2 and F are each of exponential size. Hence the sets $\mathcal{T}_1^{\natural}, \mathcal{T}_2^{\natural}, \mathcal{T}_3$ are each of doubly exponential size. Nullary predicates are also doubly exponentially many. Hence $\mathsf{S}_{\mathsf{max}}$ is of triple exponential size.

Theorem 4. *Satisfiability for the class $\mathcal{C}_1^{\oplus}$ is in 3-EXPTIME.*

Corollary 1. *Secrecy for single blind copying protocols with XOR is in 3-EXPTIME.*

5 Conclusion

The precise complexity of satisfiability for the class $\mathcal{C}_1^{\oplus}$, and of secrecy for single blind copying protocols with XOR, is still open. Both problems are DEXPTIME-complete in absence of the XOR theory. The 3-EXPTIME upper bound shown here means that the price of adding the XOR theory is low. In comparison adding any of the theories AC, ACU or AG leads to undecidability. Similar behavior is observed when studying two-way alternating tree automata, where alternation leads to undecidability in presence of theories AC, ACU and AG, but has not too high a complexity in presence of the theory XOR [18]. We also showed how the idea of decomposing one-variable terms can be generalized to the XOR case to obtain a low complexity upper bound for the satisfiability problem. This technique is of independent interest for automated deduction.

References

1. Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In: CSFW 2001, pp. 82–96. IEEE Computer Society Press, Los Alamitos (2001)
2. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 124–135. Springer, Heidelberg (2003)
3. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: An NP decision procedure for protocol insecurity with XOR. In: LICS 2003, pp. 261–270 (2003)
4. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007), `http://www.grappa.univ-lille3.fr/tata`
5. Comon-Lundh, H., Cortier, V.: New decidability results for fragments of first-order logic and application to cryptographic protocols. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 148–164. Springer, Heidelberg (2003)
6. Comon-Lundh, H., Cortier, V.: Security properties: Two agents are sufficient. In: Degano, P. (ed.) ESOP 2003. LNCS, vol. 2618, pp. 99–113. Springer, Heidelberg (2003)
7. Comon-Lundh, H., Shmatikov, V.: Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In: LICS 2003, pp. 271–280. IEEE Computer Society Press, Los Alamitos (2003)
8. Cortier, V.: Vérification Automatique des Protocoles Cryptographiques. PhD thesis, ENS Cachan, France (2003)
9. Cortier, V.: Private communication (May 2008)
10. Cortier, V., Delaune, S., Lafourcade, P.: A survey of algebraic properties used in cryptographic protocols. Journal of Computer Security 14(1), 1–43 (2006)
11. Cortier, V., Keighren, G., Steel, G.: Automatic analysis of the security of XOR-based key management schemes. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 538–552. Springer, Heidelberg (2007)
12. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory IT-29(2), 198–208 (1983)
13. Durgin, N.A., Lincoln, P., Mitchell, J., Scedrov, A.: Undecidability of bounded security protocols. In: FMSP 1999, Trento, Italy (1999)
14. Küsters, R., Truderung, T.: Reducing protocol analysis with XOR to the XOR-free case in the Horn theory based approach. In: CCS 2008, pp. 129–138. ACM Press, New York (2008)
15. Rusinowitch, M., Turuani, M.: Protocol insecurity with finite number of sessions is NP-complete. In: CSFW 2001. IEEE Computer Society Press, Los Alamitos (2001)
16. Seidl, H., Verma, K.N.: Flat and one-variable clauses: Complexity of verifying cryptographic protocols with single blind copying. ACM Transactions on Computational Logic 9(4) (2008)
17. Verma, K.N.: Two-way equational tree automata for AC-like theories: Decidability and closure properties. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 180–196. Springer, Heidelberg (2003)
18. Verma, K.N.: Alternation in equational tree automata modulo XOR. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 518–530. Springer, Heidelberg (2004)
19. Verma, K.N., Seidl, H., Schwentick, T.: On the complexity of equational Horn clauses. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS, vol. 3632, pp. 337–352. Springer, Heidelberg (2005)

Protocol Security and Algebraic Properties: Decision Results for a Bounded Number of Sessions

Sergiu Bursuc[1] and Hubert Comon-Lundh[1,2]

[1] LSV, ENS Cachan & CNRS & INRIA
[2] RCIS, AIST Tokyo

Abstract. We consider the problem of deciding the security of cryptographic protocols for a bounded number of sessions, taking into account some algebraic properties of the security primitives, for instance Abelian group properties. We propose a general method for deriving decision algorithms, splitting the task into 4 properties of the rewriting system describing the intruder capabilities: locality, conservativity, finite variant property and decidability of one-step deducibility constraints. We illustrate this method on a non trivial example, combining several Abelian Group properties, exponentiation and a homomorphism, showing a decidability result for this combination.

1 Introduction

Following the work of [7,12,16,18] and many others, for a bounded number of protocol instances (sessions), the existence of attacks on some security properties (such as secrecy) of cryptographic protocols can be expressed as the existence of solutions of *deducibility constraints*. In this setting, the messages are abstracted by terms in some quotient term algebra. The semantics of deducibility constraints depends on an equational theory, that represents the intruder capabilities and the algebraic properties of the cryptographic primitives. Given such an equational theory E and assuming a fixed (bounded) number of sessions, we may guess an interleaving of actions of the given sessions. Then the security problem can be formulated as follows:

Input: given a sequence of variables $\{x_1, \ldots, x_n\}$ and an increasing sequence of finite sets of terms $T_1 \subseteq \ldots \subseteq T_n$ (representing the increasing intruder's knowledge) such that $Var(T_i) \subseteq \{x_1, \ldots, x_{i-1}\}$, and terms $u_1, v_1, \ldots, u_n, v_n$,
Question: do there exist contexts $C_1, \ldots, C_n$ and an assignment σ such that

$$C_1[T_1]\sigma =_E x_1\sigma \wedge \ldots \wedge C_n[T_n]\sigma =_E x_n\sigma \wedge u_1\sigma =_E v_1\sigma \wedge \ldots \wedge u_n\sigma =_E v_n\sigma?$$

The equations $u_i = v_i$ represent the tests performed by the agents. The variables x_i correspond to the terms forged by the intruder; as they occur in u_i, v_i, they must pass the tests of honest agents. The contexts $C_1, \ldots, C_n$ correspond to the computations (the *recipes*) of the attacker. We will give several examples.

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 133–147, 2009.
© Springer-Verlag Berlin Heidelberg 2009

As E-unification is a special case, the decidability of E-unifiability is a necessary condition for the decidability of deducibility constraints. This is not sufficient, as, for instance, when $E = AGh$, unification modulo E is decidable and unitary, while deducibility constraints are undecidable [11].

There are several equational theories for which deducibility constraints are decidable (in NP in most cases). This is the case for the so-called Dolev-Yao theory [17], for exclusive or [4,9], for some theories partially modeling modular exponentiation [5,18], etc. There are also a few results for classes of equational theories: combination results [6,7] and monoidal theories [12]. We were faced however with a case study that does not fall into the scope of any of these general results. In this example, we need to model more properties of modular exponentiation, otherwise there is no honest execution of the protocol [3].

This particular example of application turns out to be non-trivial. We do not want however to build a new theory, only for this specific example. That is why, in the present paper, we try to draw some general method for solving deducibility constraints, in the presence of algebraic properties. We assume first that the equational theory is defined by an AC-convergent term rewriting system $\mathcal{R}$. Then we decompose the problem into 4 tasks:

Locality: if there is a context C such that $C[T] \xrightarrow[\mathcal{R}]{*} t$ and t is in normal form, then C can be chosen such that, for any subterm C' of C, $C'[T]\downarrow$ belongs to an a priori computable set of terms $\mathcal{D}(T, t)$. This yields (depending on the notion of subterm) decision algorithms for the "passive attacks" (given T, t, is there a C such that $C[T]\downarrow = t$)

Conservativity: amounts, in the Dolev-Yao case, to the "small attack property": if there is a solution to the constraint C, then there is another solution σ in which the subterms of $x\sigma$ are instances of the subterms of C. This is the core of the decidability proof of [17]. In the case of other algebraic properties, the situation is more complicated as we have to replace the syntactic notion of subterms with a semantic one [7].

Finite variants: this property of a (AC) rewrite system is a slight generalization of (AC)-basic narrowing termination. It states that instantiation and rewriting can be commuted: it is possible to anticipate possible reductions triggered by instantiations. As shown in [8], this is satisfied by most equational theories that are relevant to security protocols. There are however important exceptions, such as homomorphism properties (typically AGh).

Pure deducibility constraints: are deducibility constraints in which the contexts C_i that are applied to the left members, are restricted to be *pure*, i.e. to have only leaves as proper subterms. For a standard definition of subterms, this can occur only when C_i consists of a single function symbol. In the case of disjoint combinations [6], pure contexts consist of symbols of a single component theory .

We follow a similar approach as in [12]. However, in that paper, the authors only consider monoidal theories, together with Dolev-Yao rules. For instance, modeling modular exponentiation could not fall in the scope of [12], as (at least)

two associative-commutative symbols are necessary. The idea then would be, following [7], to use hierarchical combination results. However, our case study is not a hierarchical combination.

We borrow ideas from both papers and try to give a constraint solving procedure that can be applied to both hierarchical combinations and to our case study EP (*Electronic Purse*): both EP and the well-moded systems of [7] satisfy conservativity and EP is local. Then, together with the finite variant property, this allows us to reduce deducibility constraints to pure deducibility constraints (Section 3.4). Then we focus in the section 4 on pure deducibility constraint solving. This problem has much in common with combination problems. We do not provide however a general solution, but only consider our case study, trying to emphasize the main steps of this combination method.

Many proofs (including non trivial ones) are missing in this paper and can be found in [1].

2 Rewriting and Security

2.1 Term Rewriting

We use classical notations and terminology from [13] on terms, unification, rewrite systems. We only give here the less standard definitions. $\mathcal{F}$ is a set of function symbols with their arity. $\mathcal{F}_{ac} \subseteq \mathcal{F}$ is a set of associative and commutative (AC in short) symbols. They are considered as varyadic. The sets of terms $\mathcal{T}(\mathcal{F}, X)$ and $\mathcal{T}(\mathcal{F})$ are defined as usual, except that we assume flattening: these algebras are rather quotients by the AC congruence for some symbols. Everywhere, equality has to be understood modulo AC. Replacements of subterms may require flattening the terms. $top(t)$ is the root symbol of t.

A substitution $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ maps each variable x_i to t_i and all other variables to themselves. It is extended to an endomorphism of $\mathcal{T}(\mathcal{F}, X)$.

Many functions from terms to terms (resp. from terms to sets of terms) are extended without mention to sets of terms by $f(S) \stackrel{\text{def}}{=} \{f(t) \mid t \in S\}$ (resp. $f(S) \stackrel{\text{def}}{=} \bigcup_{t \in S} f(t)$) and to substitutions: $f(\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}) \stackrel{\text{def}}{=} \{x_1 \mapsto f(t_1), \ldots, x_n \mapsto f(t_n)\}$ (resp. $f(\{x \mapsto t_1, \ldots, x_n \mapsto t_n\}) \stackrel{\text{def}}{=} f(\{t_1, \ldots, t_n\})$).

A term rewriting system $\mathcal{R}$ is convergent modulo $AC(\mathcal{F}_{ac})$ if it is terminating and Church Rosser modulo $AC(\mathcal{F}_{ac})$. In such a case, $s{\downarrow}$ is the normal form of the term s w.r.t. $\mathcal{R}$ (and modulo AC).

We often use the notation of contexts: $C[T]$ or $\zeta[T]$ when $T = (t_1, \ldots, t_n)$ is a sequence of terms. C, ζ are actually terms whose variables belong to $\{x_1, \ldots, x_n\}$ and $C[T]$ is another notation for $C\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. The ordering on terms in T is only relevant to match the appropriate variable x_i: when we abstract the actual names x_i (as in the notation $C[T]$), we do not need to precise the ordering on terms in T and therefore, by abuse of notation, we use $C[T]$ when T is a set (and not a sequence).

2.2 A Relevant Equational Theory

We will consider the following equational theory as a guideline in the paper. The set of function symbols $\mathcal{F}_{EP} = \{exp, h, +, J_+, e_+, \star, J_\star, e_\star, \bullet, J_\bullet, e_\bullet\}$ can be interpreted in the integers: exp is the (binary) exponentiation, h is some fixed-base exponentiation, $+$ is the addition, $\star$ and $\bullet$ both represent multiplication. We carefully chose the function symbols and designed an equational theory that includes sufficiently many arithmetic properties, so that the protocol can be executed [3], yet avoiding distributivity axioms that would yield undecidability. The introduction of two function symbols for the multiplication simplifies the proofs of Sections 4, 3.2, but we do not know if it is necessary.

For each $\circ \in \{+, \star, \bullet\}$, $\mathcal{R}_{AG(\circ)}$ is the rewrite system modulo $AC(\circ)$:

$$\begin{array}{rr}
x \circ e_\circ \to x & x \circ J_\circ(x) \to e_\circ \\
J_\circ(x) \circ J_\circ(y) \to J_\circ(x \circ y) & J_\circ(e_\circ) \to e_\circ \\
J_\circ(J_\circ(x)) \to x & J_\circ(x) \circ x \circ y \to y \\
J_\circ(x) \circ J_\circ(y) \circ z \to J_\circ(x \circ y) \circ z & J_\circ(x \circ y) \circ x \to J_\circ(y) \\
J_\circ(x \circ y) \circ x \circ z \to J_\circ(y) \circ z & J_\circ(J_\circ(x) \circ y) \to x \circ J_\circ(y)
\end{array}$$

We define $\mathcal{R}_{EP} = \mathcal{R}_0 \cup \mathcal{R}_{AG(+)} \cup \mathcal{R}_{AG(\star)} \cup \mathcal{R}_{AG(\bullet)}$, where:

$$\mathcal{R}_0 = \left\{ \begin{array}{rr}
exp(h(x), y) \to h(x \star y) & J_\bullet(h(x)) \to h(J_+(x)) \\
exp(exp(x, y), z) \to exp(x, y \star z) & h(e_+) \to e_\bullet \\
h(x) \bullet h(y) \to h(x + y) & J_\bullet(h(x) \bullet y) \to h(J_+(x)) \bullet J_\bullet(y) \\
h(x) \bullet h(y) \bullet z \to h(x + y) \bullet z & exp(e_\bullet, x) \to h(e_+ \star x) \\
exp(x, e_\star) \to x &
\end{array} \right.$$

Lemma 1. *$\mathcal{R}_{EP}$ is convergent modulo $AC(+, \star, \bullet)$.*

This has been mechanically verified using CiME [10].

2.3 Semantic Subterms

Splitting a problem, for instance an equational theory, into subproblems, requires to gather together the symbols from the individual theories. This yields a notion of *semantic subterms* in which a *pure term* (resp. *pure context*) is a term t whose only subterms are its leaves and t itself. This notion must not be too rough, otherwise it does not yield any simplification. For instance, in the extreme case, if we define the semantic subterms as the leaves of a term, then every term is pure and we don't gain anything. On the other side, every rule of the rewriting system must have pure members (we have to decide to which subproblem it will belong).

A typical example of semantic subterms is given by a *mode assignment* as in [7]. We do not have space to describe it in detail here. Anyway, in our case study there is no appropriate mode assignment: they all yield a too rough notion of subterm in which, for instance, $exp(u, v)$ would not be a subterm of $exp(u, v) \bullet w$. That is why we need another (refined) definition of semantic subterms for our example.

The set of subterms of t is defined recursively from its immediate subterms (called *factors*): $\mathtt{Sub}_{EP}(t) \stackrel{\text{def}}{=} \{t\} \cup \mathtt{Sub}_{EP}(\mathrm{Fact}_{EP}(t))$. We now focus on the definition of factors.

Intuitively, the factors retrieve the "alien" subterms. The factors of both sides of the rules in $\mathcal{R}_{EP}$ must be variables. Roughly, in the following definition, we carry a state (the index of $Fact$) that memorizes which part of a member of a rule has been recognized so far. For instance $a+b$ is a factor of $(a+b) \bullet u$, since there is no rule that can break $a+b$ without removing first the $\bullet$. $a+b$ is not a factor of $h(a+b) \bullet u$ since, depending on u, this can be rewritten into a term $h(a+b+v) \bullet w$, of which $a+b$ is not a subterm.

Definition 1 (EP-Factors,$\top$). *We let $Fact_{EP}(t) = Fact_{\top(t)}(t)$ where $Fact_f$ and $\top(t)$ are defined as follows (we let $\circ$ be any symbol in $\{+, \star, \bullet\}$):*

- $\top(t) = \circ$, *if* $top(t) \in \{\circ, e_\circ, J_\circ\}$, $\top(h(t)) = \bullet$, *if* $\top(t) = +$ *and* $\top(h(t)) = exp$, *if* $\top(t) = \star$. *In all other cases,* $\top(t) = top(t)$.
- $Fact_\circ(C_\circ[t_1, \ldots, t_n]) = \bigcup_{i=1}^{n} Fact_\circ(t_i)$ *if* $C_\circ \in \mathcal{T}(\{\circ, J_\circ, e_\circ\}, \mathcal{X})$
- $Fact_\bullet(h(t)) = Fact_+(t)$ *and, in all other cases,* $Fact_\circ(t) = \{t\}$ *if* $\top(t) \neq \circ$
- $Fact_{exp}(exp(u, v)) = Fact_{exp}(u) \cup Fact_\star(v)$, $Fact_{exp}(h(u)) = Fact_\star(u)$ *and, in all other cases,* $Fact_{exp}(t) = \{t\}$ *if* $\top(t) \neq exp$
- *For other function symbols* f, $Fact_f(f(t_1, ..., t_n)) = \{t_1, \ldots, t_n\}$ *and* $Fact_f(t) = \{t\}$ *if* $top(t) \neq f$.

Example 1. $\mathrm{Fact}_{EP}(exp(h(a \star b), c \star d)) = \{a, b, c, d\}$; $\mathrm{Fact}_{EP}(h(a \star b) \bullet h(a+c) \bullet (a+d)) = \{a \star b, a, c, a+d\}$.

2.4 Deducibility Constraints

Definition 2 (Constraint systems). *A* deducibility constraint system C *is a finite set of equations S between terms of $\mathcal{T}(\mathcal{F}, X)$, together with a finite sequence of deducibility constraints $\{T_1 \stackrel{?}{\vdash} x_1, \ldots, T_n \stackrel{?}{\vdash} x_n\}$, where each T_i is a finite set of terms in $\mathcal{T}(\mathcal{F}, X)$, $x_1, \ldots, x_n$ are distinct variables and such that:*

Origination: $Var(T_i) \subseteq \{x_1, \ldots, x_{i-1}\}$, *for all* $1 \leq i < n$.
Monotonicity: $T_i \subseteq T_{i+1}$, *for all* $1 \leq i < n$.

Monotonicity expresses that an attacker never forgets any message. Origination expresses the commitment of the intruder to some message at each protocol step. Note that some variables of S may not belong to $\{x_1, \ldots, x_n\}$.

Example 2. The following is *not* a constraint system, as it violates the origination property:

$$\begin{cases} a, b \stackrel{?}{\vdash} z & z = x + y \\ a, b, x \stackrel{?}{\vdash} y & \end{cases}$$

$$\mathcal{S}_1 = \begin{cases} a \stackrel{?}{\vdash} x & x = h(y + a) \\ a, x \bullet h(b) \stackrel{?}{\vdash} y & \end{cases}$$

is a constraint system. This corresponds to an agent expecting a message that matches $h(y + a)$ and replying $h(y + a) \bullet h(b)$. Then we ask which values of y can be deduced.

All function symbols in $\mathcal{F}$ are public (i.e. available to the intruder), except some secret data, modeled by the constants from a set C_{priv} .

Definition 3 (Solution). *A* recipe *is any context $\zeta \in T(\mathcal{F} \setminus C_{priv}, X)$: this corresponds to the attacker's sequences of actions.*

A substitution σ is a solution *(resp. a* pure solution*) of the constraint system C if its domain contains all the free variables of C and*

- *for every $s = t \in S$, $s\sigma\downarrow = t\sigma\downarrow$*
- *For every $T \stackrel{?}{\vdash} s \in C$, there is a recipe (resp. a pure recipe) ζ such that $\zeta[T\sigma]\downarrow = s\sigma\downarrow$*

Example 3. Consider the system $\mathcal{S}_1$ of example 2. From $a \stackrel{?}{\vdash} x$, x is assigned a term constructed on a and public function symbols. Then, from $x = h(y + a)$, the same property must hold for y. If $b \in C_{priv}$, we cannot use $x \bullet h(b)$ in the second constraint: the solutions assign to y any term whose only private constant symbol is a and then assign $h(y\sigma + a)\downarrow$ to x.

We are interested in the satisfiability of constraints systems: "is there an attack on the protocol ?".

3 The Four Main Properties

3.1 Locality

(Generalized) Locality relies on the notion of semantic subterms and on a function $\mathcal{D}$ that may add or remove a few contexts. $\mathcal{D}$ is defined by a finite set of replacement rules $u_i \mapsto v_i$, $1 \leq i \leq n$. Then $\mathcal{D}(t) = \{v_i\theta \mid t = u_i\theta\}$.

Definition 4. *A rewriting system $\mathcal{R}$ is* local *w.r.t. $\mathcal{D}$ and $\mathtt{Sub}()$ if, for every recipe C, every finite set of terms T in normal form, there is a recipe ζ such that $\zeta[T]\downarrow = C[T]\downarrow$ and, $\forall \zeta' \in \mathtt{Sub}(\zeta)$, $\zeta'[T]\downarrow \in \mathcal{D}(\mathtt{Sub}(T)) \cup \mathcal{D}(\mathtt{Sub}(\zeta[T]\downarrow))$.*

In words: if t can be deduced from T, then there is a recipe ζ such that $\zeta[T]\downarrow = t$ and, adding the pure components of ζ one-by-one and normalizing at each step, we stay close to the subterms of T and t. The notion of "closeness" is provided by $\mathcal{D}$. Since we can compute beforehand all terms that are close to subterms of T or t, the locality can be used for solving the *passive attacker problem*: "given T, t in normal form, is there a recipe ζ such that $\zeta[T]\downarrow = t$?": we saturate by pure deducibility the set of terms that are close to a subterm of T or t.

The equational theories that are relevant to security are local [4,7,9,12]. In the EP case, we showed a locality property, for another notion of subterm [3]. Here, $\mathcal{D}_{EP}$ is defined by the replacements $\{x \mapsto x; x \mapsto h(x); h(x) \mapsto x\}$.

Lemma 2. *EP is local (w.r.t. $\mathbf{Sub}_{EP}()$ and $\mathcal{D}_{EP}$).*

Example 4. Let $T = \{h(h(a) + J_+(b)), b\}$ and let $t = h(h(a) \star b)$. The smallest recipe ζ s.t. $\zeta[T]\downarrow = t$ is $\zeta = exp(x_1 \bullet h(x_2), x_2)$. If we consider $\zeta' = x_1 \bullet h(x_2) \in \mathtt{Sub}_{EP}(\zeta)$, we get $\zeta'[T]\downarrow = h(h(a))$, which is not in $\mathtt{Sub}_{EP}(T) \cup \mathtt{Sub}_{EP}(t)$. However, $\zeta'[T]\downarrow \in \mathcal{D}_{EP}(\mathtt{Sub}_{EP}(T))$

3.2 Conservativity

Example 5. Let $T_1 = \{a, b\}$, $T_2 = \{a, b, c, x + a\}$ and consider the deducibility constraint :

$$a, b \overset{?}{\vdash} x \quad \wedge \quad a, b, c, x + a \overset{?}{\vdash} y$$

and a solution $\sigma = \{x \mapsto a \star b; y \mapsto b + c\}$. There is a recipe in normal form $\zeta = ((x_4 + J_+(x_1)) \star J_\star(x_1)) + x_3$ such that $\zeta[T_2]\sigma\downarrow = y\sigma$ (remember that x_i refers to the ith element in the list of terms):

$$\zeta[T_2]\sigma = (((x + a + J_+(a)) \star J_\star(a)) + c)\sigma \rightarrow ((x \star J_\star(a)) + c)\sigma \rightarrow b + c$$

The last redex however is an overlap between $\zeta[T_2]\downarrow$ and σ. It consists in retrieving some subterm of $x\sigma$ (here b). This is an un-necessarily complicated way of getting b: there is a simpler recipe $\zeta' = x_2 + x_3$ that would yield also $y\sigma$.

Roughly, the subterms of $x\sigma$ can be obtained from T_1, hence, by monotonicity, they can be obtained from T_2: there is a ζ' such that $\zeta'[T_2]\sigma\downarrow = y\sigma$ and such that no rewriting step retrieves a subterm from $x\sigma$. This is what conservativity formalizes. However, in general, we may need to retrieve some subterms of $x\sigma$, but only along fixed paths, specified by a function $\mathcal{D}$.

We assume again a function $\mathcal{D}$ that is defined by a finite set of replacement rules (it does not need to be the same as for locality).

Definition 5. *E is conservative (w.r.t. $\mathcal{D}$ and $\mathbf{Sub}()$) if, for any satisfiable deducibility constraint system C, there is a solution σ such that $\mathbf{Sub}(C\sigma\downarrow) \subseteq \mathcal{D}(\mathbf{Sub}(C)\sigma\downarrow)$.*

Lemma 3. *EP is conservative w.r.t. $\mathcal{D}_{EP}$ and $\mathbf{Sub}_{EP}()$.*

The proof of this lemma is non-trivial and requires several intermediate steps that we cannot display here (see [1] for more details). The main ideas are the following: assume that σ is a solution of C and $t \in \mathtt{Sub}_{EP}(x\sigma), t \notin \mathcal{D}_{EP}(\mathtt{Sub}_{EP}(C)\sigma\downarrow)$. Then we replace t with an arbitrary constant c, yielding again a solution. Roughly, if $\zeta[T]\sigma\downarrow = x\sigma$, then replacing t with c in both members is possible, and yields $\zeta[T]\sigma'\downarrow = x\sigma'$, except when t or some $u \in \mathcal{D}_{EP}(t)$ occurs in the derivation, i.e. when, for some $\zeta' \in \mathtt{Sub}(\zeta)$, $\zeta'[T]\sigma\downarrow \in \mathcal{D}_{EP}(t)$. Then we show that we can construct another recipe for the term u, in which all private constants, which are irrelevant in the derivation $\zeta\sigma\downarrow = u$, are replaced with public constants.

Example 6. $\mathcal{D}_{EP}$ is necessary in Lemma 3:

$$\mathcal{C} = \begin{cases} h(a+b) \bullet c & \overset{?}{\vdash} x \\ h(a+b) \bullet c, exp(x \bullet J_{\bullet}(c), c \star d) & \overset{?}{\vdash} y \qquad y = h(z \star d) \end{cases}$$

$\sigma = \{x \mapsto h(a+b) \bullet c, y \mapsto h((a+b) \star c \star d), z \mapsto (a+b) \star c\}$ is a solution. $t = a + b \in \mathtt{Sub}_{EP}(\mathcal{C}\sigma\downarrow)$. $t \in \mathcal{D}_{EP}(\mathtt{Sub}_{EP}(\mathcal{C}))\sigma\downarrow \backslash \mathtt{Sub}_{EP}(\mathcal{C})\sigma\downarrow$

The proof given in [1] actually covers classes of equational theories and semantic subterms, of which EP and well-moded systems of [7] are instances.

3.3 Finite Variant Property

This notion is introduced in [8]. It states that possible reductions on instances of t can be anticipated: they are instances of the *finite variants of* t. A typical example in which we get finite variants are rewrite systems for which (basic) narrowing terminates. More examples are available in [8,14].

Definition 6. *An AC-convergent rewrite system R has* the finite variant property*, if, for any term t, there is a finite set of terms $V(t) = \{t\theta_1\downarrow, \ldots, t\theta_k\downarrow\}$ such that $\forall\sigma.\exists\theta, \exists u \in V(t).t\sigma\downarrow = u\theta$.*

The monoidal theories do not have necessarily this property: an equation $h(x+y) = h(x)+h(y)$ cannot be oriented in any way that yields a finite set of variants for both $h(x)$ and $x+y$. In [7] there is an hypothesis called "reducibility of the theory", that is similar to (but weaker than) the finite variant property.

Lemma 4. *EP has the finite variant property.*

Example 7. Consider the term $x + J_+(a)$. Its finite variants (for EP) are $e_+ = (a + J_+(a))\downarrow$, $y = (y + a + J_+(a))\downarrow$,$J_+(a) = (e_+ + J_+(a))\downarrow$, $J_+(y+a) = (J_+(y) + J_+(a))\downarrow$, $x + J_+(a)$. Note that orienting the inverse rule in the other direction, though yielding an AC-convergent rewrite system for Abelian group, would not yield the finite variant property, as noticed in [8].

3.4 A Decision Algorithm for Deducibility Constraints

A *Pure deducibility constraint* is a deducibility constraint, of which only pure solutions are considered (see Definition 3).

Theorem 1. *If E is local, conservative and has the finite variant property, then the satisfiability of deducibility constraints is reducible to the satisfiability of pure deducibility constraints.*

Proof sketch: We assume w.l.o.g. that the functions $\mathcal{D}$ for conservativity and locality are identical (this may require merging the two sets of replacements).

Let C be the constraint system and σ be a solution of C. By conservativity, C has a solution σ_1 such that $\mathtt{Sub}(C\sigma_1\downarrow) \subseteq \mathcal{D}(\mathtt{Sub}(C)\sigma_1\downarrow)$. Thanks to the finite

variant property, there is a variant $C\theta_1\downarrow$ such that $\mathtt{Sub}(C)\sigma_1\downarrow \subseteq \mathtt{Sub}(C\theta_1\downarrow)\sigma_2$ for some substitution σ_2 such that $\sigma_1 = \theta_1\sigma_2$. Then C is satisfiable iff some system $CS_1 = C\theta_1\downarrow$ has a solution σ_2 such that $\mathtt{Sub}(C)\sigma_1\downarrow \subseteq \mathcal{D}(\mathtt{Sub}(CS_1)\sigma_2)$.

Now, we pull the substitution out of the scope of $\mathcal{D}$ as follows. If $\mathcal{D}$ is defined by the replacements $u_i \mapsto v_i$, we guess which patterns u_i might be introduced by the substitution σ_2. Let θ_2 be a substitution such that, for every x, $x\theta_2$ is a renaming of some term in $\mathtt{Sub}(u_1, \ldots, u_n)$. C is satisfiable iff some $CS_2 = CS_1\theta_2$ has a solution σ_3 such that $\mathcal{D}(\mathtt{Sub}(CS_1)\sigma_2) \subseteq \mathcal{D}(\mathtt{Sub}(CS_2))\sigma_3$.

By locality, the intermediate steps in the proofs can be assumed to belong to $\mathcal{D}(\mathtt{Sub}(C\sigma_1\downarrow)) \subseteq \mathcal{D}(\mathcal{D}(\mathtt{Sub}(CS_2))\sigma_3)$. As before, we pull out σ_3: there is a substitution θ_3, and $CS_3 = CS_2\theta_3$ such that $\mathcal{D}(\mathtt{Sub}(C\sigma_1\downarrow)) \subseteq \mathcal{D}(\mathcal{D}(\mathtt{Sub}(CS_3)))\sigma_4$, for some σ_4. Then, we non-deterministically choose $\theta = \theta_1\theta_2\theta_3$, add to the equational part of C the equations $x = x\theta$ for each variable of C. Then, we guess which terms in $\mathcal{D}(\mathcal{D}(\mathtt{Sub}(CS_3)))$ are deducible, and in which order: we insert some deducibility constraints (iteratively) replacing $T_i \stackrel{?}{\vdash} x_i \ \wedge \ T_{i+1} \stackrel{?}{\vdash} x_{i+1}$ with a system

$$T_i \stackrel{?}{\vdash} x_i \ \wedge \ T_i \stackrel{?}{\vdash} z_1 \ \wedge \ T_i, z_1 \stackrel{?}{\vdash} z_2 \ \wedge \ldots \wedge \ T_{i+1}, z_1, \ldots, z_m \stackrel{?}{\vdash} x_{i+1}$$

where $z_1, \ldots, z_n$ are new variables and $z_1 = v_1 \wedge \ldots \wedge z_m = v_m$ is added to the equational part, for the guessed $v_1, \ldots, v_m \in \mathcal{D}(\mathcal{D}(\mathtt{Sub}(CS_3)))$.

By locality, any solution of the original system can be extended to the new variables into a *pure* solution of the resulting system. □

Example 8. Consider the deducibility constraint $(a, b, c \in C_{priv})$ in EP:

$$C = \begin{cases} a \star b & \stackrel{?}{\vdash} x \\ a \star b, exp(x, c), J_\star(b \star c), h(a) + c & \stackrel{?}{\vdash} y \end{cases}$$

$\sigma = \{x \mapsto h(a \star b); y \mapsto a\}$ is a solution. Let us see the branch of the above transformation yielding a pure constraint of which an extension of σ is a solution.

First, we compute a variant, guessing that $x = h(z)$ and the normal form of $exp(x, c)$ is $h(z \star c)$. Next, we guess the identity for θ_2, θ_3 and we guess which terms in $\{a, b, c, z, h(a), h(b), h(c), h(z)\} \subseteq \mathcal{D}(\mathcal{D}(\mathtt{Sub}_{EP}(CS_3)))$ are deducible and in which order. For instance we get the pure system:

$$C' = \begin{cases} a \star b & \stackrel{?}{\vdash} x & x \ = h(z) \\ a \star b, exp(x, c), J_\star(b \star c), h(a) + c & \stackrel{?}{\vdash} z_1 & z_1 = h(a) \\ a \star b, exp(x, c), J_\star(b \star c), h(a) + c, z_1 & \stackrel{?}{\vdash} z_2 & z_2 = c \\ a \star b, exp(x, c), J_\star(b \star c), h(a) + c, z_1, z_2 & \stackrel{?}{\vdash} y & \end{cases}$$

$\sigma' = \sigma \uplus \{z_1 \mapsto h(a); z_2 \mapsto c\}$ is a pure solution of C': we use the pure recipes $exp(x_2, x_3)$, $x_4 + J_+(x_5)$, $x_1 * J_\star(x_3 \star x_6)$ to obtain respectively $z_1\sigma', z_2\sigma', y\sigma'$. (Each time x_i is replaced with the ith term in T_j)

4 Pure Deducibility Constraints

In this section, we consider only the case study EP. We first fix the pure recipes in this case. Then, we can guess, for each elementary constraint, which type of recipe is used and we rely on a combination method.

W.r.t. classical combination methods, there are two additional difficulties. First, we cannot introduce new variables for abstracting subterms, as we might lose the origination property of the constraints, without which pure deducibility constraints become undecidable [2]. Second, the theories are not disjoint nor hierarchical: we cannot carelessly generate constraints in theory 2 while solving constraints in theory 1. Compared with [7], we do not have a bound on the number of steps performed in a higher theory.

Instead of abstracting with new variables, we show in Lemma 6 that, when $\top(u\sigma\downarrow) \neq \top(u)$, there must be another, strictly smaller (w.r.t. a well chosen ordering $\geq$), term v in $\mathcal{D}_{EP}(\mathtt{Sub}_{EP}(C))$ such that $u\sigma\downarrow = v\sigma\downarrow$. Then we add $u = v$ to the equational part of the constraint and replace u with v everywhere. Moreover, $\geq$ is defined in such a way that the resulting system still satisfies origination. After successive replacements, we "stabilize" the root symbol, allowing to fix the alien subterms in each individual constraint.

The rest of this section is devoted to the proof of the following theorem. The complexity can be derived from a careful analysis of each step.

Theorem 2. *For the equational theory EP, pure deducibility constraints can be decided in NP.*

4.1 Reduction to Three Recipe Types

We consider 3 basic types of deducibility constraints: $T \overset{?}{\vdash}_\circ x$ for $\circ \in \{+, \star, \bullet\}$. A solution of such a constraint is a substitution σ such that

- if $\circ \in \{+, \star\}$, then there is a $\zeta_\circ \in \mathcal{T}(\{\circ, J_\circ, e_\circ\}, \mathcal{X})$ such that $\zeta_\circ[T]\sigma\downarrow = x\sigma$
- if $\circ = \bullet$, then there exist $\zeta_\bullet \in \mathcal{T}(\{\bullet, J_\bullet, e_\bullet\}, \mathcal{X}), \zeta_+ \in \mathcal{T}(\{+, J_+, e_+\}, \mathcal{X})$ such that $\zeta_\bullet[T]\sigma \bullet h(\zeta_+[T])\sigma\downarrow = x\sigma$.

We further restrict the pure deducibility constraints, reducing again the relevant recipes. Let a *basic deducibility constraint* be a conjunction of equations and of a deducibility constraint $T_1 \overset{?}{\vdash}_{\circ_1} x_1, \ldots, T_n \overset{?}{\vdash}_{\circ_n} x_n$ (still satisfying origination and monotonicity).

Lemma 5. *For any deducibility constraint C, we can effectively compute a finite set of basic deducibility constraints $C_1, \ldots, C_n$ such that the set of solutions of C is the union of the solutions of $C_1, \ldots, C_n$.*

To prove this, we first note that there are only 6 possible pure recipe types and reduce three of them to basic deducibility constraints.

Example 9. Let $a, b \overset{?}{\vdash} x$ and consider the (non-basic) pure recipe $exp(x_1, \zeta_\star)$. Replacing x_1 with a, we get the basic constraint $a, b \overset{?}{\vdash}_\star y \wedge x = exp(a, y)$. Similarly, if the pure recipe is $h(\zeta_\circ)$ $(\circ \in \{+, \star\})$, we get $a, b \overset{?}{\vdash}_\circ y \wedge x = h(y)$.

In addition, we will split $\overset{?}{\vdash}_\bullet : T \overset{?}{\vdash}_\bullet x$ becomes $[T; T] \overset{?}{\vdash}_\bullet x$ and, by definition, σ is a solution of $[T_1; T_2] \overset{?}{\vdash}_\bullet x$ if there are pure recipes $\zeta^1_\bullet$ and ζ^2_+ such that $\zeta^1_\bullet(T_1\sigma) \bullet h(\zeta^2_+(T_2\sigma))\downarrow = x\sigma$.

4.2 Guessing Top Symbols and Equalities

We wish to guess the head function symbol of a term (after instantiation and normalization). Such guesses are recorded using additional constraints $H(u) \in S_t$, where S_t is either a finite set of function symbols or "Others". We also write $H(u) = f$ instead of $H(u) \in \{f\}$.

A substitution σ satisfies $H(u) \in S_t$ if $\top(u\sigma\downarrow) \in S_t.\sigma$ satisfies $H(u) \in$ "*Others*" if $u\sigma\downarrow$ is a constant, not occurring in $\mathcal{R}_{EP}$.

For each variable x in C, we guess a constraint $H(x) = f$. By abuse of notation, we let $\top(x)$ be f when the constraint system contains $H(x) = f$.

As in combination procedures, we also guess all equalities between terms in $\mathcal{D}_{EP}(\mathtt{Sub}_{EP}(C))$. Such guesses are recorded by adding equalities to the equational part of the constraint. After this step, we may only consider solutions σ such that, for any $u, v \in \mathcal{D}_{EP}(\mathtt{Sub}_{EP}(C))$, if $u\sigma\downarrow = v\sigma\downarrow$, then $u =_{Eq(C)} v$: all identities that are triggered by the substitution applications are already consequences of the equational part $Eq(C)$ of C.

4.3 Stabilizing the Root Symbol

If $x, y \in Var(C)$, we let $x \succ_C y$ if, for every constraint $T \overset{?}{\vdash} x \in C$, y is a variable of T. This defines an ordering $\succeq_C$, thanks to the origination property. It is extended to symbols of $\mathcal{F}$ by $x \succ_C f$ for $x \in Var(C), f \in \mathcal{F}$ and $f \succ_C h$ for $f \in \mathcal{F} \setminus \{h\}$. Then $\geq$ is the multiset path ordering [13] on the precedence $\succeq_C$.

When the top symbol of a subterm is not stable by substitution and normalization, it equals a smaller subterm of the system:

Lemma 6. *For every $u \in \mathcal{Sub}_{EP}(C)$ and every solution θ, if $\top(u\theta\downarrow) \neq \top(u)$, there is a $v \in \mathcal{D}_{EP}(\mathcal{Sub}_{EP}(C))$ s.t. $v < u$ and $u\theta\downarrow = v\theta\downarrow$.*

Now, we perform the following transformation: for every $u \in \mathtt{Sub}_{EP}(C)$ such that $u =_{Eq(C)} t$ and $t \in \mathcal{D}_{EP}(\mathtt{Sub}_{EP}(C))$ and $u > t$, replace u with t in the left sides of deducibility constraints. Since equalities have been guessed, by lemma 6, after iterating the above replacements, the root symbols are stable, and the resulting system is still a constraint system, thanks to the definition of $\geq$.

Example 10.

$$\left.\begin{array}{ll} a \bullet b, c & \overset{?}{\vdash}_{+} x \\ a \bullet b, c, x + J_{+}(c) & \overset{?}{\vdash}_{\bullet} y \end{array}\right\} \Rightarrow \left\{\begin{array}{l} a \bullet b, c \overset{?}{\vdash}_{+} x \\ a \bullet b, c \overset{?}{\vdash}_{\bullet} y \end{array}\right.$$

Assume $x\sigma = a \bullet b + c$. Then the equality $x + J_{+}(c) = a \bullet b$ is part of the equations of C and, since $x + J_{+}(c) > a \bullet b$, the transformation yields the system on the right.

4.4 Eliminating Variables from Left Hand Sides: Reducing Deducibility Constraints to Linear Diophantine Equations

At this stage, we could consider the alien subterms in the T_i's as constants and translate the deducibility constraints into Diophantine systems. This yields however non-linear Diophantine equations, as shown in [2]: we need to rely on origination and the Abelian group properties for further simplifications.

Let $T_i \overset{?}{\vdash}_{\circ} x_i$ be a constraint of C. In this step, we eliminate variables from $Fact_{\circ}(T_i)$, if they are headed with $\circ$ and introduced by a $\circ$ constraint.[1]

Let $\circ \in \{+, \star, \bullet\}$, $T_i \overset{?}{\vdash}_{\circ} x_i$ (resp. $[T_i, T_i] \overset{?}{\vdash}_{\bullet} x_i$) in C. We define $\mathsf{fv}^{\circ}_i(u)$ as follows: we let $\mathcal{X}^{\circ}_i$ be the set of variables x_j, $j < i$ such that $\top(x_j) = \circ$ and $T_j \overset{?}{\vdash}_{\circ} x_j \in C$ (resp. $[T_j, T_j] \overset{?}{\vdash}_{\bullet} x_j \in C$) and $u = \zeta_{\circ}[x_{i_1}, \ldots, x_{i_k}, u_1, \ldots, u_m]$ be such that $\mathrm{Fact}_{\circ}(u) \cap \mathcal{X}^{\circ}_i = \{x_{i_1}, \ldots, x_{i_k}\}$. Then $\mathsf{fv}^{\circ}_i(u) \overset{\text{def}}{=} \zeta_{\circ}[e_{\circ}, \ldots, e_{\circ}, u_1, \ldots, u_m]{\downarrow}$.

If $\circ \in \{+, \star\}$ and $T_i \overset{?}{\vdash}_{\circ} x_i \in C$, we replace every $u \in T_i$ with $\mathsf{fv}^{\circ}_i(u)$. If $\circ = \bullet$ and $[T_i, T_i] \overset{?}{\vdash}_{\bullet} x_i \in C$, we replace any $u = u' \bullet h(u'')$ in the first copy of T_i with $\mathsf{fv}^{\bullet}_i(u') \bullet h(\mathsf{fv}^{+}_i(u''))$ and every u in the second copy with $\mathsf{fv}^{+}_i(u)$.

Example 11. An instance of the above transformation is:

$$a \overset{?}{\vdash} x \quad \wedge \quad a, b + x \overset{?}{\vdash} y \quad \Longrightarrow \quad a \overset{?}{\vdash} x \quad \wedge \quad a, b \overset{?}{\vdash} y$$

This preserves the solutions: if ζ_{+} is such that $\zeta_{+}[a, b + x\sigma]{\downarrow} = y\sigma$, then, rebuilding $x\sigma$ from a, we can build a new recipe ζ'_{+} such that $\zeta'_{+}[a, b] = y\sigma$. Conversely, we can build ζ_{+} from ζ'_{+} by subtracting $x\sigma$ when necessary. These ideas were already applied to deduction constraints modulo Abelian groups in [18].

Similarly $x_1 \bullet a \bullet h(x_2 + b)$ would be replaced with $a \bullet h(b)$, if $\top(x_1) = \bullet$ and $\top(x_2) = +$.

The left hand sides of deducibility constraints might no longer be linearly ordered by inclusion, but we no longer need this property in further sections.

Lemma 7. *The above transformations preserve the solutions and result in a basic constraint C' such that:*

[1] We can prove actually that the variables that are headed with $\circ$ and not introduced by a $\circ$ constraint have been eliminated in the previous step.

- *if* $T_i \overset{?}{\vdash}_\circ x_i \in C'$ *and* $\circ \in \{+, \star\}$, *then for every* $x \in Fact_\circ(T_i)$, $\top(x) \neq \circ$.
- *if* $[T_i', T_i''] \overset{?}{\vdash}_\bullet x_i \in C'$, *then:*
  - *for every* $x \in Fact_+(T_i'')$, $\top(x) \neq +$.
  - *for every* $u = u_1 \bullet \ldots u_n \bullet h(v_1 + \ldots + v_m) \in T_i'$ *such that* $Fact_\bullet(u) = \{u_1, \ldots, u_n, v_1, \ldots, v_m\}$, *for every* $x \in \mathcal{X}$, $x \in \{u_1, \ldots, u_n\} \Rightarrow \top(x) \neq \bullet$ *and* $x \in \{v_1, \ldots, v_m\} \Rightarrow \top(x) \neq +$.

4.5 Turning Deduction Constraints into Linear Diophantine Equations

We do the following transformation of deducibility constraints into equations. This transformation is possible and correct by lemmas 6 and 7. For $\circ \in \{+, \star\}$,

$$\sum_i t_i^1, \ldots, \sum_i t_i^n \overset{?}{\vdash}_\circ x \quad \Longrightarrow \quad x = \sum_{i,j} \lambda_j t_i^j$$

if, $\forall i, j. \top(t_i^j) \neq \circ$ and $\lambda_1, \ldots, \lambda_n$ are new formal integer variables, representing the number of times each term is selected in the recipe.

For $\overset{?}{\vdash}_\bullet$, we use here a multiplicative notation for $\bullet$ and an additive one for $+$:

$$[\ (\textstyle\prod_i t_i^1) \bullet h(\sum_i u_i^1),\ \ldots,\ (\prod_i t_i^n) \bullet h(\sum_i u_i^n)\ ;\ \sum_i v_i^1, \ldots, \sum_i v_i^m\ ] \overset{?}{\vdash} x$$
$$\Rightarrow \begin{cases} x = x_1 \bullet h(x_2) \\ x_1 = \prod_{j=1}^n \prod_i (t_i^j)^{\lambda_j} \\ x_2 = \sum_{j=1}^n \sum_i \lambda_j u_i^j + \sum_{j=1}^m \sum_i \mu_j v_i^j \end{cases}$$

If $\forall i, j. \top(t_i^j) \notin \{\bullet, h\}$ & $\top(u_i^j), \top(v_i^j) \neq +$ and $\lambda_1, \ldots, \lambda_n, \mu_1, \ldots, \mu_m$ are integer variables.

Example 12.

$$[\ a \bullet h(2b),\ a^3 \bullet b\ ;\ 2a + 3b, 2b\ ] \overset{?}{\vdash}_\bullet x$$

is turned into $x = x_1 \bullet h(x_2)$, $x_1 = a^{\lambda_1} \bullet a^{3\lambda_2} \bullet b^{\lambda_2} = a^{\lambda_1 + 3\lambda_2} \bullet b^{\lambda_2}$, $x_2 = 2\lambda_1 b + 2\mu_1 a + 3\mu_1 b + 2\mu_2 b = 2\mu_1 a + (2\lambda_1 + 3\mu_1 + 2\mu_2)b$.

4.6 Solving the System of Equations

We now have to solve a system of equations $E = E_1 \cup E_2$, where E_1 contains equations of the form $x = \Sigma_\circ \lambda_i \alpha_i t_i, H(t_i) \neq \circ$ and E_2 is a set of usual equations. After applying the finite variant property, our procedure for solving E is similar to unification procedures in the union of disjoint theories, except that we have in addition the linear Diophantine equations coming from the deducibility constraints. We recall here very briefly the main steps.

Step 1: apply the finite variant property. Equations modulo EP are reduced to equations in a combination of 3 AC theories.

Step 2: guess equalities, theories and an occurrence ordering. Since new equalities can be introduced in step 1, we guess once more the equalities between the subterms of E.

Step 3: turn the system into linear diophantine equations. The equations modulo AC yield linear Diophantine systems. That is where constraints of E_1 are inserted. After the above steps, equations in E_1 are of the form:

$$\beta_1 u_1 \circ \ldots \circ \beta_m u_m = \lambda_1 \alpha_1 t'_1 \circ \ldots \circ \lambda_n \alpha_n t'_n,\ \forall i. H(t'_i) \neq \circ$$

Since equalities have been guessed, we can simplify the equations and turn it into a linear system. For instance, if $\circ = +$ and $u_i = \sum_{j=1}^{n} u_j^i t'_j$:

$$\alpha_1 \lambda_1 = \beta_1 \mu_1^1 + \ldots + \beta_m \mu_1^m + \gamma_1 \ \wedge\ \ldots\ \wedge\ \alpha_n \lambda_n = \beta_1 \mu_n^1 + \ldots + \beta_m \mu_n^m + \gamma_n$$

5 Conclusion

We gave a general method for deciding deducibility constraints in the presence of algebraic properties of security primitives and apply it to a non-trivial example: from Theorems 1 and 2, we deduce:

Theorem 3. *In the case of the equational theory EP, deducibility constraints are decidable (in NP).*

There is still much to do. First, we need to undestand better the combination mechanism of section 4. We chose to explain the steps of the procedure in some detail here since we feel that there should be some more general mechanism, that would be applicable to other combination problems. Next, though we have general conditions on the semantic subterms and on the rewriting systems, that imply conservativity, we did not display these conditions here, since we feel that there should be simpler conditions. For instance, it should be possible, from the rewrite system, to automatically infer the definition of subterms, in such a way that we get conservativity. This is an ambitious goal, as it would yield a systematic way of splitting an equational theory into (non disjoint, yet combinable) equational theories.

Acknowledgements

We thank Stéphanie Delaune for her numerous comments and contributions to an earlier draft of this work, as well as the anonymous referees for their valuable suggestions.

References

1. Bursuc, S., Comon-Lundh, H.: Protocols, insecurity decision and combination of equational theories. Technical Report 02, Laboratoire Spécification et Vérification (February 2009), `http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2009-02.pdf`

2. Bursuc, S., Comon-Lundh, H., Delaune, S.: Associative-commutative deducibility constraints. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 634–645. Springer, Heidelberg (2007)
3. Bursuc, S., Comon-Lundh, H., Delaune, S.: Deducibility constraints, equational theory and electronic money. In: Comon-Lundh, H., Kirchner, C., Kirchner, H. (eds.) Jouannaud Festschrift. LNCS, vol. 4600, pp. 196–212. Springer, Heidelberg (2007)
4. Chevalier, Y., Kuester, R., Rusinowitch, M., Turuani, M.: An NP decision procedure for protocol insecurity with xor. In: Kolaitis [15]
5. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 124–135. Springer, Heidelberg (2003)
6. Chevalier, Y., Rusinowitch, M.: Combining Intruder Theories. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 639–651. Springer, Heidelberg (2005)
7. Chevalier, Y., Rusinowitch, M.: Hierarchical combination of intruder theories. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 108–122. Springer, Heidelberg (2006)
8. Comon-Lundh, H., Delaune, S.: The finite variant property: How to get rid of some algebraic properties. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 294–307. Springer, Heidelberg (2005)
9. Comon-Lundh, H., Shmatikov, V.: Intruder deductions, constraint solving and insecurity decision in preence of exclusive or. In: Kolaitis [15]
10. Contejean, E., Marché, C.: Cime: Completion modulo e. In: Ganzinger, H. (ed.) RTA 1996. LNCS, vol. 1103, pp. 416–419. Springer, Heidelberg (1996)
11. Delaune, S.: An undecidability result for AGh. Theoretical Computer Science 368(1-2), 161–167 (2006)
12. Delaune, S., Lafourcade, P., Lugiez, D., Treinen, R.: Symbolic protocol analysis for monoidal equational theories. Information and Computation 206(2-4), 312–351 (2008)
13. Dershowitz, N., Jouannaud, J.-P.: Rewrite systems. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 243–309. North-Holland, Amsterdam (1990)
14. Escobar, S., Meseguer, J., Sasse, R.: Effectively checking the finite variant property. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 79–93. Springer, Heidelberg (2008)
15. Kolaitis, P. (ed.): Eighteenth Annual IEEE Symposium on Logic in Computer Science, Ottawa, Canada. IEEE Computer Society, Los Alamitos (2003)
16. Millen, J., Shmatikov, V.: Constraint solving for bounded-process cryptographic protocol analysis. In: Proc. 8th ACM Conference on Computer and Communications Security (2001)
17. Rusinowitch, M., Turuani, M.: Protocol insecurity with finite number of sessions is np-complete. In: Proc. 14th IEEE Computer Security Foundations Workshop, Cape Breton, Nova Scotia (June 2001)
18. Shmatikov, V.: Decidable analysis of cryptographic protocols with products and modular exponentiation. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 355–369. Springer, Heidelberg (2004)

YAPA: A Generic Tool for Computing Intruder Knowledge*

Mathieu Baudet[1], Véronique Cortier[2], and Stéphanie Delaune[3]

[1] DCSSI, France
[2] LORIA, CNRS & INRIA project Cassis, France
[3] LSV, ENS Cachan & CNRS & INRIA, France

Abstract. Reasoning about the knowledge of an attacker is a necessary step in many formal analyses of security protocols. In the framework of the applied pi calculus, as in similar languages based on equational logics, knowledge is typically expressed by two relations: deducibility and static equivalence. Several decision procedures have been proposed for these relations under a variety of equational theories. However, each theory has its particular algorithm, and none has been implemented so far.

We provide a generic procedure for deducibility and static equivalence that takes as input any convergent rewrite system. We show that our algorithm covers all the existing decision procedures for convergent theories. We also provide an efficient implementation, and compare it briefly with the more general tool ProVerif.

1 Introduction

Understanding security protocols often requires reasoning about the information accessible to an online attacker. Accordingly, many formal approaches to security rely on a notion of *deducibility* [18,19] that models whether a piece of data, typically a secret, is retrievable from a finite set of messages. Deducibility, however, does not always suffice to reflect the knowledge of an attacker. Consider for instance a protocol sending an encrypted Boolean value, say, a vote in an electronic voting protocol. Rather than deducibility, the key idea to express confidentiality of the plaintext is that an attacker should not be able to *distinguish* between the sequences of messages corresponding to each possible value.

In the framework of the applied pi-calculus [3], as in similar languages based on equational logics [10], indistinguishability corresponds to a relation called *static equivalence*: roughly, two sequences of messages are *statically equivalent* when they satisfy the same algebraic relations from the attacker's point of view. Static equivalence plays an important role in the study of guessing attacks (e.g. [1,5,13]), as well as for anonymity properties and electronic voting protocols (e.g. [17]). In several cases, this notion has also been shown to imply the more complex and precise notion of cryptographic indistinguishability [1,8], related to probabilistic polynomial-time Turing machines.

* Kindly supported by ANR-07-SESU-002 AVOTÉ and ARA SSIA FormaCrypt.

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 148–163, 2009.
© Springer-Verlag Berlin Heidelberg 2009

We emphasize that both deducibility and static equivalence apply to observations on finite sets of messages, and do not take into account the dynamic behavior of protocols. Nevertheless, deducibility is used as a subroutine by many general decision procedures [12,11]. Besides, it has been shown that observational equivalence in the applied pi-calculus coincides with labeled bisimulation [3], that is, corresponds to checking a number of static equivalences and some standard bisimulation conditions.

Deducibility and static equivalence rely on an underlying equational theory for axiomatizing the properties of cryptographic functions. Many decision procedures [2,14] have been proposed to compute these relations under a variety of equational theories, including symmetric and asymmetric encryptions, signatures, exclusive OR, and homomorphic operators. However, except for the class of subterm convergent theories [2], which covers the standard flavors of encryption and signature, each of these decision results introduces a new procedure, devoted to a particular theory. Even in the case of the general decidability criterion given in [2], we note that the algorithm underlying the proof has to be adapted for each theory, depending on how the criterion is fulfilled.

Perhaps as a consequence of this fact, none of these decision procedures has been implemented so far. Up to our knowledge the only tool able to verify static equivalence is ProVerif [9,10]. This general tool can handle various equational theories and analyze security protocols under active adversaries. However termination of the verifier is not guaranteed in general, and protocols are subject to (safe) approximations.

The present work aims to fill this gap between theory and implementation and propose an efficient tool for deciding deducibility and static equivalence in a uniform way. It is initially inspired from a procedure for solving more general constraint systems related to active adversaries and equivalence of finite processes, presented in [5], with corrected extended version in [6] (in French). However, due to the complexity of the constraint systems, this decision procedure was only studied for subterm convergent theories, and remains too complex to enable an efficient implementation.

Our first contribution is to provide and study a generic procedure for checking deducibility and static equivalence, taking as input any convergent theory (that is, any equational theory described by a finite convergent rewrite system). We prove the algorithm sound and complete, up to explicit failure cases. Note that (unfailing) termination cannot be guaranteed in general since the problem of checking deducibility and static equivalence is undecidable, even for convergent theories [2]. To address this issue and turn our algorithm into a decision procedure for a given convergent theory, we provide two criteria. First, we define a syntactic criterion on the rewrite rules that ensures that the algorithm never fails. This criterion is enjoyed in particular by any convergent subterm theory, as well as the theories of blind signature and homomorphic encryption. Termination often follows from a simple analysis of the rules of the algorithm: as a proof of concept, we obtain a new decidability result for deducibility and static equivalence for the prefix theory, representing encryption in CBC mode.

Second, we provide a termination criterion based on deducibility: provided that failure cannot occur, termination on a given input is equivalent to the existence of some natural finite representation of deducible terms. As a consequence, we obtain that our algorithm can decide deducibility and static equivalence for all the convergent theories previously known to be decidable [2].

Our second contribution is an efficient implementation of this generic procedure, called YAPA. After describing the main features of the implementation, we report several experiments suggesting that our tool computes static equivalence faster and for more convergent theories than the general tool ProVerif [9,10]. Because of space constraints, proofs are in an extended version of this paper [7].

2 Preliminaries

2.1 Term Algebra

We start by introducing the necessary notions to describe cryptographic messages in a symbolical way. For modeling cryptographic primitives, we assume a given set of *function symbols* $\mathcal{F}$ together with an arity function $\mathrm{ar} : \mathcal{F} \to \mathbb{N}$. Symbols in $\mathcal{F}$ of arity 0 are called *constants*. We consider a set of *variables* $\mathcal{X}$ and a set of additional constants $\mathcal{W}$ called *parameters*. The (usual, first-order) term algebra generated by $\mathcal{F}$ over $\mathcal{W}$ and $\mathcal{X}$ is written $\mathcal{F}[\mathcal{W} \cup \mathcal{X}]$ with elements denoted by $T, U, T_1 \ldots$ More generally, we write $\mathcal{F}'[A]$ for the least set of terms containing a set A and stable by application of symbols in $\mathcal{F}' \subseteq \mathcal{F}$.

We write $\mathrm{var}(T)$ (resp. $\mathrm{par}(T)$) for the set of variables (resp. parameters) that occur in a term T. These notations are extended to tuples and sets of terms in the usual way. The set of positions (resp. subterms) of a term T is written $\mathrm{pos}(T) \subseteq \mathbb{N}^*$ (resp. $\mathrm{st}(T)$). The subterm of T at position $p \in \mathrm{pos}(T)$ is written $T|_p$. The term obtained by replacing $T|_p$ with a term U in T is denoted $T[U]_p$.

A *(finite, partial) substitution* σ is a mapping from a finite subset of variables, called its *domain* and written $\mathrm{dom}(\sigma)$, to terms. The *image* of a substitution is its image as a mapping $\mathrm{im}(\sigma) = \{\sigma(x) \mid x \in \mathrm{dom}(\sigma)\}$. Substitutions are extended to endomorphisms of $\mathcal{F}[\mathcal{X} \cup \mathcal{W}]$ as usual. We use a postfix notation for their application. A term T (resp. a substitution σ) is *ground* iff $\mathrm{var}(T) = \emptyset$ (resp. $\mathrm{var}(\mathrm{im}(\sigma)) = \emptyset$).

For our cryptographic purposes, it is useful to distinguish a subset $\mathcal{F}_{\mathsf{pub}}$ of $\mathcal{F}$, made of *public function symbols*, that is, intuitively, the symbols made available to the attacker. A *recipe* (or *second-order term*) M, N, $M_1 \ldots$ is a term in $\mathcal{F}_{\mathsf{pub}}[\mathcal{W} \cup \mathcal{X}]$, that is, a term containing no *private* (non-public) function symbols. A *plain term* (or *first-order term*) t, r, s, $t_1 \ldots$ is a term in $\mathcal{F}[\mathcal{X}]$, that is, containing no parameters. A *(public, ground, non-necessarily linear) n-ary context* C is a recipe in $\mathcal{F}_{\mathsf{pub}}[\mathsf{w}_1, \ldots, \mathsf{w}_n]$, where we assume a fixed countable subset of parameters $\{\mathsf{w}_1, \ldots, \mathsf{w}_n, \ldots\} \subseteq \mathcal{W}$. If C is a n-ary context, $C[T_1, \ldots, T_n]$ denotes the term obtained by replacing each occurrence of w_i with T_i in C.

2.2 Rewriting

A *rewrite system* $\mathcal{R}$ is a finite set of *rewrite rules* $l \to r$ where $l, r \in \mathcal{F}[\mathcal{X}]$ and $\mathrm{var}(r) \subseteq \mathrm{var}(l)$. A term S *rewrites* to T by $\mathcal{R}$, denoted $S \to_{\mathcal{R}} T$, if there exist $l \to r$ in $\mathcal{R}$, $p \in \mathrm{pos}(S)$ and a substitution σ such that $S|_p = l\sigma$ and $T = S[r\sigma]_p$. We write $\to_{\mathcal{R}}^{+}$ for the transitive closure of $\to_{\mathcal{R}}$, $\to_{\mathcal{R}}^{*}$ for its reflexive and transitive closure, and $=_{\mathcal{R}}$ for its reflexive, symmetric and transitive closure.

A rewrite system $\mathcal{R}$ is *convergent* if is *terminating*, i.e. there is no infinite chains $T_1 \to_{\mathcal{R}} T_2 \to_{\mathcal{R}} \ldots$, and *confluent*, i.e. for every terms S, T such that $S =_{\mathcal{R}} T$, there exists U such that $S \to_{\mathcal{R}}^{*} U$ and $T \to_{\mathcal{R}}^{*} U$.

A term T is $\mathcal{R}$*-reduced* if there is no term S such that $T \to_{\mathcal{R}} S$. If $T \to_{\mathcal{R}}^{*} S$ and S is $\mathcal{R}$-reduced then S is *a* $\mathcal{R}$*-reduced form of* T. When this reduced form is unique (in particular if $\mathcal{R}$ is convergent), we write $S = T{\downarrow_{\mathcal{R}}}$.

2.3 Equational Theories

We equip the signature $\mathcal{F}$ with an equational theory represented by a set of equations $\mathcal{E}$ of the form $s = t$ with $s, t \in \mathcal{F}[\mathcal{X}]$. The equational theory E generated by $\mathcal{E}$ is the least set of equations containing $\mathcal{E}$ that is stable under the axioms of congruence (reflexivity, symmetry, transitivity, application of function symbols) and under application of substitutions. We write $=_{\mathsf{E}}$ for the corresponding relation on terms. Equational theories have proved very useful for modeling algebraic properties of cryptographic primitives [2,15].

We are particularly interested in theories E that can be represented by a convergent rewrite system $\mathcal{R}$, i.e. theories for which there exists a convergent rewrite system $\mathcal{R}$ such that the two relations $=_{\mathcal{R}}$ and $=_{\mathsf{E}}$ coincide. The rewrite system $\mathcal{R}$ —and by extension the equational theory E— is *subterm convergent* if, in addition, we have that for every rule $l \to r \in \mathcal{R}$, r is either a subterm of l or a ground $\mathcal{R}$-reduced term. This class encompasses the one of the same name used in [2], the class of dwindling theories used in [4], and the class of public-collapsing theories introduced in [16].

Example 1. Consider the signature $\mathcal{F}_{\mathsf{enc}} = \{\mathsf{dec}, \mathsf{enc}, \langle _, _ \rangle, \pi_1, \pi_2\}$. The symbols dec, enc and $\langle _, _ \rangle$ are functional symbols of arity 2 that represent respectively the decryption, encryption and pairing functions, whereas π_1 and π_2 are functional symbols of arity 1 that represent the projection function on the first and the second component of a pair, respectively. The equational theory of pairing and symmetric (deterministic) encryption, denoted by $\mathsf{E}_{\mathsf{enc}}$, is generated by the equations $\mathcal{E}_{\mathsf{enc}} = \{\mathsf{dec}(\mathsf{enc}(x, y), y) = x,\ \ \pi_1(\langle x, y \rangle) = x,\ \ \pi_2(\langle x, y \rangle) = y\}$.

Motivated by the modeling of the ECB mode of encryption, we may also consider an encryption symbol that is homomorphic with respect to pairing:

$$\mathcal{E}_{\mathsf{hom}} = \mathcal{E}_{\mathsf{enc}} \cup \left\{ \begin{array}{l} \mathsf{enc}(\langle x, y \rangle, z) = \langle \mathsf{enc}(x, z), \mathsf{enc}(y, z) \rangle \\ \mathsf{dec}(\langle x, y \rangle, z) = \langle \mathsf{dec}(x, z), \mathsf{dec}(y, z) \rangle \end{array} \right\}.$$

If we orient the equations from left to right, we obtain two rewrite systems $\mathcal{R}_{\mathsf{enc}}$ and $\mathcal{R}_{\mathsf{hom}}$. Both rewrite systems are convergent, only $\mathcal{R}_{\mathsf{enc}}$ is subterm convergent.

From now on, we assume a given equational theory E represented by a convergent rewrite system $\mathcal{R}$. A symbol f is *free* if f does not occur in $\mathcal{R}$. In order to model (an unbounded number of) random values possibly generated by the attacker, we assume that $\mathcal{F}_{\mathsf{pub}}$ contains infinitely many free public constants. We will use free private constants to model secrets, for instance the secret keys used to encrypt a message. Private (resp. public) free constants are closely related to bound (resp. free) *names* in the framework of the applied pi calculus [3]. Our formalism also allows one to consider non-constant private symbols.

3 Deducibility and Static Equivalence

In order to describe the cryptographic messages observed or inferred by an attacker, we introduce the following notions of deduction facts and frames.

A *deduction fact* is a pair, written $M \rhd t$, made of a recipe $M \in \mathcal{F}_{\mathsf{pub}}[\mathcal{W} \cup \mathcal{X}]$ and a plain term $t \in \mathcal{F}[\mathcal{X}]$. Such a deduction fact is *ground* if $\mathrm{var}(M, t) = \emptyset$. A *frame*, denoted by letters φ, Φ, $\Phi_0 \ldots$, is a finite set of ground deduction facts. The *image* of a frame is defined by $\mathrm{im}(\Phi) = \{t \mid M \rhd t \in \Phi\}$. A frame Φ is *one-to-one* if $M_1 \rhd t$, $M_2 \rhd t \in \Phi$ implies $M_1 = M_2$.

A frame φ is *initial* if it is of the form $\varphi = \{w_1 \rhd t_1, \ldots, w_\ell \rhd t_\ell\}$ for some distinct parameters $w_1, \ldots, w_\ell \in \mathcal{W}$. Initial frames are closely related to the notion of frames in the applied pi-calculus [3]. The parameters w_i can be seen as labels that refer to the messages observed by an attacker. Given such an initial frame φ, we denote by $\mathrm{dom}(\varphi)$ its *domain* $\mathrm{dom}(\varphi) = \{w_1, \ldots, w_\ell\}$. If $\mathrm{par}(M) \subseteq \mathrm{dom}(\varphi)$, we write $M\varphi$ for the term obtained by replacing each w_i by t_i in M. If in addition M is ground then $t = M\varphi$ is a ground plain term.

3.1 Deducibility, Recipes

Classically (see e.g. [2]), a ground term t is *deducible* modulo E from an initial frame φ if there exists $M \in \mathcal{F}_{\mathsf{pub}}[\mathrm{dom}(\varphi)]$ such that $M\varphi =_{\mathsf{E}} t$. This corresponds to the intuition that the attacker may compute (infer) t from φ. For the purpose of our study, we generalize this notion to arbitrary frames, and even sets of (non-necessarily ground) deduction facts ϕ, using the notations $\rhd_\phi$ and $\rhd_\phi^{\mathsf{E}}$.

Definition 1 (deducibility). *Let ϕ be finite set of deductions facts, for instance a frame. We say that M* is a recipe of t in ϕ, *written $M \rhd_\phi t$, iff there exist a (public, ground, non-necessarily linear) n-ary context C and some deduction facts $M_1 \rhd t_1$, …, $M_n \rhd t_n$ in ϕ such that $M = C[M_1, \ldots, M_n]$ and $t = C[t_1, \ldots, t_n]$. In that case, we say that t is* syntactically deducible *from ϕ, also written $\phi \vdash t$.*

We say that M is a recipe of t in ϕ modulo E, *written $M \rhd_\phi^{\mathsf{E}} t$, iff there exists a term t' such that $M \rhd_\phi t'$ and $t' =_{\mathsf{E}} t$. In that case, we say that t is* deducible from ϕ modulo E, *written $\phi \vdash_{\mathsf{E}} t$.*

We note that $M \rhd_\varphi t$ is equivalent to $M\varphi = t$ when φ is an initial frame and when t (or equivalently M) is ground.

Example 2. Consider the equational theory $\mathsf{E}_{\mathsf{enc}}$ given in Example 1. Let $\varphi = \{\mathsf{w}_1 \rhd \langle \mathsf{enc}(\mathsf{s}_1, \mathsf{k}), \mathsf{enc}(\mathsf{s}_2, \mathsf{k})\rangle, \mathsf{w}_2 \rhd \mathsf{k}\}$ where s_1, s_2 and k are private constant symbols. We have that $\langle \mathsf{w}_2, \mathsf{w}_2\rangle \rhd_\varphi \langle \mathsf{k}, \mathsf{k}\rangle$, and $\mathsf{dec}(\mathsf{proj}_1(\mathsf{w}_1), \mathsf{w}_2) \rhd_\varphi^{\mathsf{E}_{\mathsf{enc}}} \mathsf{s}_1$.

3.2 Static Equivalence, Visible Equations

Deducibility does not always suffice for expressing the knowledge of an attacker. In particular, it does not account for the partial information that an attacker may obtain about secrets. This issue motivates the study of visible equations and static equivalence [3], defined as follows.

Definition 2 (static equivalence). *Let φ be an initial frame. The set of* visible equations of φ modulo E *is defined as*

$$\mathrm{eq}_{\mathsf{E}}(\varphi) = \{M \bowtie N \mid M, N \in \mathcal{F}_{\mathsf{pub}}[\mathrm{dom}(\varphi)],\ M\varphi =_{\mathsf{E}} N\varphi\}$$

where $\bowtie$ is a dedicated commutative symbol. Two initial frames φ_1 and φ_2 with the same domain are statically equivalent *modulo E, written $\varphi_1 \approx_{\mathsf{E}} \varphi_2$, if their sets of visible equations are equal, i.e.* $\mathrm{eq}_{\mathsf{E}}(\varphi_1) = \mathrm{eq}_{\mathsf{E}}(\varphi_2)$.

This definition is in line with static equivalence in the applied pi calculus [3]. For the purpose of finitely describing the set of visible equations $\mathrm{eq}_{\mathsf{E}}(\varphi)$ of an initial frame, we introduce *quantified equations* of the form $\forall z_1, \ldots, z_q.M \bowtie N$ where $z_1, \ldots, z_q \in \mathcal{X}$, $q \geq 0$ and $\mathrm{var}(M, N) \subseteq \{z_1, \ldots, z_q\}$. In the following, finite sets of quantified equations are denoted Ψ, Ψ_0,... We write $\Psi \models M \bowtie N$ when the ground equation $M \bowtie N$ is a consequence of Ψ in the usual, first-order logics with equality axioms for the relation $\bowtie$ (that is, reflexivity, symmetry, transitivity and compatibility with symbols in $\mathcal{F}_{\mathsf{pub}}$). When no confusion arises, we may refer to quantified equations simply as *equations*. As usual, quantified equations are considered up to renaming of bound variables.

Example 3. Consider again the equational theory $\mathsf{E}_{\mathsf{enc}}$ given in Example 1. Let $\varphi_1 = \{\mathsf{w}_1 \rhd \mathsf{enc}(\mathsf{c}_0, \mathsf{k}),\ \mathsf{w}_2 \rhd \mathsf{k}\}$ and $\varphi_2 = \{\mathsf{w}_1 \rhd \mathsf{enc}(\mathsf{c}_1, \mathsf{k}),\ \mathsf{w}_2 \rhd \mathsf{k}\}$ where c_0, c_1 are public constants and k is a private constant. Let $\Psi_1 = \{\mathsf{enc}(\mathsf{c}_0, \mathsf{w}_2) \bowtie \mathsf{w}_1\}$ and $\Psi_2 = \{\mathsf{enc}(\mathsf{c}_1, \mathsf{w}_2) \bowtie \mathsf{w}_1\}$. We have that $\Psi_i \models \mathrm{eq}_{\mathsf{E}_{\mathsf{enc}}}(\varphi_i)$ for $i = 1, 2$. Hence, $\mathrm{eq}_{\mathsf{E}_{\mathsf{enc}}}(\varphi_1) \neq \mathrm{eq}_{\mathsf{E}_{\mathsf{enc}}}(\varphi_2)$ and the two frames φ_1 and φ_2 are not statically equivalent. However, it can be shown that $\{\mathsf{w}_1 \rhd \mathsf{enc}(\mathsf{c}_0, \mathsf{k})\} \approx_{\mathsf{E}_{\mathsf{enc}}} \{\mathsf{w}_1 \rhd \mathsf{enc}(\mathsf{c}_1, \mathsf{k})\}$.

4 Main Procedure

In this section, we describe our algorithms for checking deducibility and static equivalence on convergent rewrite systems. After some additional notations, we present the core of the procedure, which consists of a set of transformation rules used to saturate a frame and a finite set of quantified equations. We then show how to use this procedure to decide deducibility and static equivalence, provided that saturation succeeds.

Soundness and completeness of the saturation procedure are detailed in Section 5. We provide sufficient conditions on the rewrite systems to ensure success of saturation in Section 6.

4.1 Decompositions of Rewrite Rules

Before stating the procedure, we introduce the following notion of *decomposition* to account for the possible superpositions of an attacker's context with a left-hand side of rewrite rule.

Definition 3 (decomposition). *Let n, p, q be non-negative integers. A (n, p, q)-*decomposition *of a term l (and by an extension of any rewrite rule $l \to r$) is a (public, ground, non-necessarily linear) context $D \in \mathcal{F}_{\mathsf{pub}}[\mathcal{W}]$ such that* $\mathrm{par}(D) = \{\mathsf{w}_1, \ldots, \mathsf{w}_{n+p+q}\}$ *and* $l = D[l_1, \ldots, l_n, y_1, \ldots, y_p, z_1, \ldots, z_q]$ *where*

- $l_1, \ldots, l_n$ *are mutually-distinct non-variable terms,*
- $y_1, \ldots, y_p$ *and* $z_1, \ldots, z_q$ *are mutually-distinct variables, and*
- $y_1, \ldots, y_p \in \mathrm{var}(l_1, \ldots, l_n)$ *whereas* $z_1, \ldots, z_q \notin \mathrm{var}(l_1, \ldots, l_n)$.

A decomposition D is proper *if it is not a parameter (i.e. $D \neq \mathsf{w}_1$).*

Example 4. Consider the rewrite rule $\mathsf{dec}(\mathsf{enc}(x, y), y) \to x$. This rule admits two proper decompositions up to permutation of parameters:

- $D_1 = \mathsf{dec}(\mathsf{enc}(\mathsf{w}_1, \mathsf{w}_2), \mathsf{w}_2)$ where $n = 0$, $p = 0$, $q = 2$, $z_1 = x$, $z_2 = y$;
- $D_2 = \mathsf{dec}(\mathsf{w}_1, \mathsf{w}_2)$ where $n = 1$, $p = 1$, $q = 0$, $l_1 = \mathsf{enc}(x, y)$ and $y_1 = y$.

4.2 Transformation Rules

To check deducibility and static equivalence, we proceed by saturating an initial frame, adding some deduction facts and equations satisfied by the frame. We consider *states* that are either the failure state $\bot$ or a couple (Φ, Ψ) formed by a one-to-one frame Φ in $\mathcal{R}$-reduced form and a finite set of quantified equations Ψ.

Given an initial frame φ, our procedure starts from an initial state associated to φ, denoted by $\mathrm{Init}(\varphi)$, obtained by reducing φ and replacing duplicated terms by equations. Formally, $\mathrm{Init}(\varphi)$ is the result of a procedure recursively defined as follows: $\mathrm{Init}(\emptyset) = (\emptyset, \emptyset)$, and assuming $\mathrm{Init}(\varphi) = (\Phi, \Psi)$, we have

$$\mathrm{Init}(\varphi \uplus \{w \rhd t\}) = \begin{cases} (\Phi, \Psi \cup \{w \bowtie w'\}) & \text{if there exists some } w' \rhd t{\downarrow_{\mathcal{R}}} \in \Phi \\ (\Phi \cup \{w \rhd t{\downarrow_{\mathcal{R}}}\}, \Psi) & \text{otherwise.} \end{cases}$$

The main part of our procedure consists in saturating a state (Φ, Ψ) by means of the transformation rules described in Figure 1. The **A** rules are designed for applying a rewrite step on top of existing deduction facts. If the resulting term is already syntactically deducible then a corresponding equation is added (rule **A.1**); or else if it is ground, the corresponding deduction fact is added to the state (rule **A.2**); otherwise, the procedure may fail (rule **A.3**). The **B** rules are meant to add syntactically deducible subterms (rule **B.2**) or related equations (rule **B.1**). For technical reasons, rule **A.1** is parametrized by a function Ctx with values of the form M or $\bot$, and satisfying the following properties:

(a) if $\phi \vdash t{\downarrow_{\mathcal{R}}}$, then for any Ψ and α, $\mathrm{Ctx}(\phi \vdash^{?}_{\mathcal{R}} t, \Psi, \alpha) \neq \bot$;

A. Inferring deduction facts and equations by context reduction

Assume that

$l = D[l_1, \ldots, l_n, y_1, \ldots, y_p, z_1, \ldots, z_q]$ is a proper decomposition of $(l \to r) \in \mathcal{R}$
$M_1 \rhd t_1, \ldots, M_{n+p} \rhd t_{n+p} \in \Phi$
$(l_1, \ldots, l_n, y_1, \ldots, y_p)\,\sigma = (t_1, \ldots, t_{n+p})$

1. If there exists $M = \mathrm{Ctx}(\Phi \cup \{z_1 \rhd z_1, \ldots, z_q \rhd z_q\} \vdash^?_{\mathcal{R}} r\sigma, \Psi, (l, r, D, \sigma))$, then

$$(\Phi, \Psi) \Longrightarrow (\Phi, \Psi \cup \{\forall z_1, \ldots, z_q.D[M_1, \ldots, M_{n+p}, z_1 \ldots, z_q] \bowtie M\}) \qquad \textbf{(A.1)}$$

2. Else, if $(r\sigma)\downarrow_{\mathcal{R}}$ is ground, then

$$(\Phi, \Psi) \Longrightarrow (\Phi \cup \{M_0 \rhd (r\sigma)\downarrow_{\mathcal{R}}\}, \Psi \cup \{\forall z_1, \ldots, z_q.D[M_1, \ldots, M_{n+p}, z_1 \ldots, z_q] \bowtie M_0\}) \qquad \textbf{(A.2)}$$

   where $M_0 = D[M_1, \ldots, M_{n+p}, \mathsf{a}, \ldots, \mathsf{a}]$ for some fixed public constant a.

3. Otherwise, $(\Phi, \Psi) \Longrightarrow \bot$ **(A.3)**

B. Inferring deduction facts and equations syntactically

Assume that $M_0 \rhd t_0, \ldots, M_n \rhd t_n \in \Phi \qquad t = f(t_1, \ldots, t_n) \in \mathrm{st}(t_0) \qquad f \in \mathcal{F}_{\mathsf{pub}}$

1. If there exists M such that $(M \rhd t) \in \Phi$,

$$(\Phi, \Psi) \Longrightarrow (\Phi, \Psi \cup \{f(M_1, \ldots, M_n) \bowtie M\}) \qquad \textbf{(B.1)}$$

2. Otherwise, $(\Phi, \Psi) \Longrightarrow (\Phi \cup \{f(M_1, \ldots, M_n) \rhd t\}, \Psi)$ **(B.2)**

Fig. 1. Transformation rules

(b) if $M = \mathrm{Ctx}(\phi \vdash^?_{\mathcal{R}} t, \Psi, \alpha)$ then there exist M' and s such that $\Psi \models M \bowtie M'$, $M' \rhd_\phi s$ and $t \to^*_{\mathcal{R}} s$. (This justifies the notation $\phi \vdash^?_{\mathcal{R}} t$ used to denote a specific deducibility problem.)

Note that a simple choice for $\mathrm{Ctx}(\phi \vdash^?_{\mathcal{R}} t, \Psi, \alpha)$ is to solve the deducibility problem $\phi \vdash^? t\downarrow_{\mathcal{R}}$ in the empty equational theory, and then return a corresponding recipe M, if any. (This problem is easily solved by induction on $t\downarrow_{\mathcal{R}}$.) Yet, optimizing the function Ctx is a nontrivial task: on the one hand, letting $\mathrm{Ctx}(\phi \vdash^?_{\mathcal{R}} t, \Psi, \alpha) \neq \bot$ for more values ϕ, t, Ψ, α makes the procedure more likely to succeed; on the other hand, it is computationally more demanding. We explain in Section 6.1 the choice of Ctx made in our implementation.

We write $\Longrightarrow^*$ for the transitive and reflexive closure of $\Longrightarrow$. The definitions of Ctx and of the transformation rules ensure that whenever $S \Longrightarrow^* S'$ and S is a state, then S' is also a state, with the same parameters unless $S' = \bot$.

Example 5. Consider the frame φ_1 previously described in Example 3. We can apply rule **A.1** as follows. Consider the rewrite rule $\mathsf{dec}(\mathsf{enc}(x, y), y) \to x$, the decomposition D_2 given in Example 4 and $t_1 = \mathsf{enc}(\mathsf{c}_0, \mathsf{k})$. We have $\mathrm{Init}(\varphi_1) = (\varphi_1, \emptyset) \Longrightarrow (\varphi_1, \{\mathsf{dec}(\mathsf{w}_1, \mathsf{w}_2) \bowtie \mathsf{c}_0\})$. In other words, since we know the key k through w_2, we can check that the decryption of w_1 by w_2 leads to the public

constant c_0. Next we apply rule **B.1** as follows: $(\varphi_1, \{\mathsf{dec}(\mathsf{w}_1, \mathsf{w}_2) \bowtie \mathsf{c}_0\}) \Longrightarrow (\varphi_1, \{\mathsf{dec}(\mathsf{w}_1, \mathsf{w}_2) \bowtie \mathsf{c}_0, \mathsf{enc}(\mathsf{c}_0, \mathsf{w}_2) \bowtie \mathsf{w}_1\})$. No more rules can then modify the state.

Main theorem. We now state the soundness and the completeness of the transformation rules provided that a *saturated state* is reached, that is, a state $S \neq \bot$ such that $S \Longrightarrow S'$ implies $S' = S$. The technical lemmas involved in the proof are detailed in Section 5.

Theorem 1 (soundness and completeness). *Let* E *be an equational theory generated by a convergent rewrite system* $\mathcal{R}$*. Let* φ *be an initial frame and* (Φ, Ψ) *be a saturated state such that* $\mathrm{Init}(\varphi) \Longrightarrow^* (\Phi, \Psi)$*.*

1. *For all* $M \in \mathcal{F}_{\mathsf{pub}}[\mathrm{par}(\varphi)]$ *and* $t \in \mathcal{F}[\emptyset]$*, we have*
$$M\varphi =_{\mathsf{E}} t \quad \Leftrightarrow \quad \exists N,\ \Psi \models M \bowtie N \text{ and } N \rhd_{\Phi} t{\downarrow_{\mathcal{R}}}$$
2. *For all* $M, N \in \mathcal{F}_{\mathsf{pub}}[\mathrm{par}(\varphi) \cup \mathcal{X}]$*, we have that* $M\varphi =_{\mathsf{E}} N\varphi \Leftrightarrow \Psi \models M \bowtie N$*.*

While the saturation procedure is sound and complete, it may not terminate, or *fail* if rule **A.3** becomes the only applicable rule. In Section 6, we explore several sufficient conditions to prevent failure and ensure termination.

4.3 Application to Deduction and Static Equivalence

Decision procedures for deduction and static equivalence follow from Theorem 1.

Algorithm for deduction. Let φ be an initial frame and t be a ground term. The procedure for checking $\varphi \vdash_{\mathsf{E}} t$ runs as follows:

1. Apply the transformation rules to obtain (if any) a saturated state (Φ, Ψ) such that $\mathrm{Init}(\varphi) \Longrightarrow^* (\Phi, \Psi)$;
2. Return *yes* if there exists N such that $N \rhd_{\Phi} t{\downarrow_{\mathcal{R}}}$ (that is, the $\mathcal{R}$-reduced form of t is syntactically deducible from Φ); otherwise return *no*.

Algorithm for static equivalence. Let φ_1 and φ_2 be two initial frames. The procedure for checking $\varphi_1 \approx_{\mathsf{E}} \varphi_2$ runs as follows:

1. Apply the transformation rules to obtain (if possible) two saturated states (Φ_1, Ψ_1) and (Φ_2, Ψ_2) such that $\mathrm{Init}(\varphi_i) \Longrightarrow^* (\Phi_i, \Psi_i)$, $i = 1, 2$;
2. For $\{i, j\} = \{1, 2\}$, for every equation $(\forall z_1, \ldots, z_\ell . M \bowtie N)$ in Ψ_i, check that $M\varphi_j =_{\mathsf{E}} N\varphi_j$ — that is, in other words, $(M\varphi_j){\downarrow_{\mathcal{R}}} = (N\varphi_j){\downarrow_{\mathcal{R}}}$;
3. If so return *yes*; otherwise return *no*.

5 Soundness and Completeness of the Saturation

The proof of Theorem 1 is based on three main lemmas. First, the transformation rules are sound in the sense that, along the saturation process, we add only deducible terms and valid equations with respect to the initial frame.

Lemma 1 (soundness). *Let φ be an initial frame and (Φ, Ψ) be a state such that* $\mathrm{Init}(\varphi) \Longrightarrow^* (\Phi, \Psi)$. *Then, we have that*

1. $M \rhd_\Phi t \;\Rightarrow\; M\varphi =_{\mathsf{E}} t$ *for all* $M \in \mathcal{F}_{\mathsf{pub}}[\mathrm{dom}(\varphi)]$ *and* $t \in \mathcal{F}[\emptyset]$;
2. $\Psi \models M \bowtie N \;\Rightarrow\; M\varphi =_{\mathsf{E}} N\varphi$ *for all* $M, N \in \mathcal{F}_{\mathsf{pub}}[\mathrm{dom}(\varphi) \cup \mathcal{X}]$.

The next two lemmas are dedicated to the completeness of **B** and **A** rules, respectively. Lemma 2 ensures that saturated states account for all the syntactic equations possibly visible. Lemma 3 deals with the reduction of a deducible term along the rewrite system $\mathcal{R}$. Using that $\mathcal{R}$ is convergent, this allows us to prove that every deducible term from a saturated frame is syntactically deducible.

Lemma 2 (completeness, syntactic equations). *Let (Φ, Ψ) be a state, and M, N be two terms such that $M \rhd_\Phi t$ and $N \rhd_\Phi t$ for some term t. Then there exists (Φ', Ψ') such that $(\Phi, \Psi) \Longrightarrow^* (\Phi', \Psi')$ using **B** rules and $\Psi' \models M \bowtie N$.*

Lemma 3 (completeness, context reduction). *Let (Φ, Ψ) be a state and M, t, t' be three terms such that $M \rhd_\Phi t$ and $t \to_{\mathcal{R}} t'$. Then, either $(\Phi, \Psi) \Longrightarrow^* \bot$ or there exist (Φ', Ψ'), M' and t'' such that $(\Phi, \Psi) \Longrightarrow^* (\Phi', \Psi')$, $M' \rhd_{\Phi'} t''$ with $t' \to^*_{\mathcal{R}} t''$, and $\Psi' \models M \bowtie M'$.*

*Besides, in both cases, the corresponding derivation from (Φ, Ψ) can be chosen to consist of a number of **B** rules, possibly followed by one instance of **A** rule involving the same rewrite rule $l \to r$ as the rewrite step $t \to_{\mathcal{R}} t'$.*

6 Termination and Non-failure

In the previous section, we proved that saturated frames yield sound and complete characterizations of deducible terms and visible equations of their initial frames. Yet, the saturation procedure may still not terminate, or fail due to rule **A.3**. In this section, we study different conditions on the rewrite system $\mathcal{R}$ so that failure never happens and/or termination is ensured.

6.1 A Syntactic Criterion to Prevent Failure

Our first criterion is syntactic and ensures that the algorithm never fails. It is enjoyed by a large class of equational theories, called *layered convergent*.

Definition 4 (layered rewrite system). *A rewrite system $\mathcal{R}$, and by extension its equational theory E, are* layered *if there exists an ascending chain of subsets $\emptyset = \mathcal{R}_0 \subseteq \mathcal{R}_1 \subseteq \ldots \subseteq \mathcal{R}_{N+1} = \mathcal{R}$ $(N \geq 0)$, such that for every $0 \leq i \leq N$, for every rule $l \to r$ in $\mathcal{R}_{i+1} - \mathcal{R}_i$, for every (n, p, q)-decomposition $l = D[l_1, \ldots, l_n, y_1, \ldots, y_p, z_1, \ldots, z_q]$, one of the following two conditions holds:*

(i) $\mathrm{var}(r) \subseteq \mathrm{var}(l_1, \ldots, l_n)$;
(ii) *there exist* $C_0, C_1, \ldots, C_k$ *and* $s_1, \ldots, s_k$ *such that*
- $r = C_0[s_1, \ldots, s_k]$;
- *for each* $1 \leq i \leq k$, $C_i[l_1, \ldots, l_n, y_1, \ldots, y_p, z_1, \ldots, z_q]$ *rewrites to* s_i *in zero or one step of rewrite rule in head position along* $\mathcal{R}_i$.

In the latter case, we say that the context $C = C_0[C_1, \ldots, C_k]$ *is* associated *to the decomposition* D *of* $l \to r$. *Note that* $C[l_1, \ldots, l_n, y_1, \ldots, y_p, z_1, \ldots, z_q] \to^*_{\mathcal{R}_i} r$.

Proposition 1. *Assume that the function* Ctx *in use is* maximal*: for every* ϕ *and* t, *if there exists* s *such that* $\phi \vdash s$ *and* $t \to^*_{\mathcal{R}} s$, *then for any* Ψ, α, $\mathrm{Ctx}(\phi \vdash^?_{\mathcal{R}} t, \Psi, \alpha) \neq \bot$. *Then, provided that* $\mathcal{R}$ *is layered convergent, there exists no state* (Φ, Ψ) *from which* $(\Phi, \Psi) \Longrightarrow \bot$ *is the only applicable derivation.*

Practical considerations. Unfortunately, such a maximal Ctx is too inefficient in practice as one has to consider the syntactic deducibility problem $\phi \vdash^? s$ for every $t \to^*_{\mathcal{R}} s$. This is why we rather use the following lighter implementation:

- for every index $0 \leq i \leq N$, and every rule $l \to r$ in $\mathcal{R}_{i+1} - \mathcal{R}_i$, if $l = D[l_1, \ldots, l_n, y_1, \ldots, y_{p+q}]$ is a (n, p, q)-decomposition satisfying condition (ii) above for some (arbitrarily chosen) associated context C, then, for every ϕ and σ such that $\phi \vdash l\sigma$, we let

$$\mathrm{Ctx}(\phi \vdash^?_{\mathcal{R}} r\sigma,\, \Psi,\, (l, r, D, \sigma)) \,=\, C[M_1, \ldots, M_{n+p+q}]$$

  where the M_k are fixed recipes such that $(M_i \rhd l_i\sigma) \in \phi$ for $1 \leq i \leq n$ and $(M_{n+j} \rhd y_j\sigma) \in \phi$ for $1 \leq j \leq p + q$;
- otherwise, if $\phi \vdash t{\downarrow_{\mathcal{R}}}$, we let $\mathrm{Ctx}(\phi \vdash^?_{\mathcal{R}} t, \Psi, \alpha)$ be some fixed M such that $M \rhd_\phi t{\downarrow_{\mathcal{R}}}$;
- in any other case, we let $\mathrm{Ctx}(\phi \vdash^?_{\mathcal{R}} t, \Psi, \alpha) = \bot$.

Using similar ideas as for the proof of Proposition 1, we can show that, for any convergent rewrite system $\mathcal{R}$, this choice of Ctx is compatible with property (b) of Subsection 4.2, and more generally with completeness, as long as, during saturation, the transformation rules **A** involve the rewrite rules of $\mathcal{R}_i$ with greater priority than those of $\mathcal{R}_j$, $i < j$. Moreover, when $\mathcal{R}$ is additionally layered, this definition ensures that the procedure never fails. Indeed, using the notations of Figure 1, $\mathrm{Ctx}(\Phi \cup \{z_1 \rhd z_1, \ldots, z_q \rhd z_q\} \vdash^?_{\mathcal{R}} r\sigma, \Psi, (l, r, D, \sigma)) = \bot$ implies that (ii) is false on D, thus (i) $\mathrm{var}(r) \subseteq \mathrm{var}(l_1, \ldots, l_n)$ holds and $(r\sigma){\downarrow_{\mathcal{R}}}$ is ground.

Example 6. Any convergent subterm rewrite system $\mathcal{R}$ is layered convergent. Indeed, let $N = 0$ and $\mathcal{R}_1 = \mathcal{R}$. For any $l \to r$ in $\mathcal{R}$ and for every decomposition $l = D[l_1, \ldots, l_n, y_1, \ldots, y_p, z_1, \ldots, z_q]$, the term r is a subterm of l, thus either $r = C[l_1, \ldots, l_n, y_1, \ldots, y_p, z_1, \ldots, z_q]$ for some context C, or r is a subterm of some l_i thus $\mathrm{var}(r) \subseteq \mathrm{var}(l_1, \ldots, l_n)$.

Example 7. Other examples are provided by the theory of homomorphism $\mathsf{E}_{\mathsf{hom}}$ defined in Section 2.3 as well as the convergent theories of blind signatures $\mathsf{E}_{\mathsf{blind}}$ and prefix encryption $\mathsf{E}_{\mathsf{pref}}$ defined by the following sets of equations.

$$\mathcal{E}_{\mathsf{blind}} = \mathcal{E}_{\mathsf{enc}} \cup \left\{ \begin{array}{c} \mathsf{unblind}(\mathsf{blind}(x,y),y) = x \\ \mathsf{unblind}(\mathsf{sign}(\mathsf{blind}(x,y),z),y) = \mathsf{sign}(x,z) \end{array} \right\}$$

$$\mathcal{E}_{\mathsf{pref}} = \mathcal{E}_{\mathsf{enc}} \cup \left\{ \mathsf{pref}(\mathsf{enc}(\langle x,y\rangle,z)) = \mathsf{enc}(x,z) \right\}$$

The theory $\mathsf{E}_{\mathsf{blind}}$ models primitives used in e-voting protocols [17]. The prefix theory represents the property of many chained modes of encryption (e.g. CBC) where an attacker can retrieve any encrypted prefix out of a ciphertext.

Let us check for instance that the prefix theory $\mathsf{E}_{\mathsf{pref}}$ is layered. Let $N = 1$, $\mathcal{R}_1$ be the rewrite system obtained from $\mathcal{E}_{\mathsf{enc}}$ by orienting the equations from left to right, and $\mathcal{R}_2 = \mathcal{R}_1 \cup \{\mathsf{pref}(\mathsf{enc}(\langle x,y\rangle,z)) \to \mathsf{enc}(x,z)\}$. The rewrite rules of $\mathcal{R}_1$ satisfy the assumptions since $\mathcal{R}_1$ forms a convergent subterm rewrite system. The additional rule $\mathsf{pref}(\mathsf{enc}(\langle x,y\rangle,z)) \to \mathsf{enc}(x,z)$ admits three decompositions up to permutation of parameters:

- $l = \mathsf{pref}(l_1)$, in which case $\mathrm{var}(r) \subseteq \mathrm{var}(l_1)$;
- $l = \mathsf{pref}(\mathsf{enc}(l_1,z))$, in which case $\mathsf{enc}(\pi_1(l_1),z) \to_{\mathcal{R}_1} r$;
- $l = \mathsf{pref}(\mathsf{enc}(\langle x,y\rangle,z))$, in which case $r = \mathsf{enc}(x,z)$.

Verifying that the convergent theories $\mathsf{E}_{\mathsf{hom}}$ and $\mathsf{E}_{\mathsf{blind}}$ are layered is similar.

6.2 Termination

In the previous subsection, we described a sufficient criterion for non-failure. To obtain decidability for a given layered convergent theory, there remains only to provide a termination argument. Such an argument is generally easy to develop by hand as we illustrate on the example of the prefix theory. For the case of existing decidability results from [2], such as the theories of blind signature and homomorphic encryption, we also provide a semantic criterion that allows us to directly conclude termination of the procedure.

Proving termination by hand. To begin with, we note that **B** rules always terminate after a polynomial number of steps. Let us write $\stackrel{\cdot}{\Longrightarrow}^n$ for the relation made of exactly n *strict applications* of rules ($S \stackrel{\cdot}{\Longrightarrow} S'$ iff $S \Longrightarrow S'$ and $S \neq S'$).

Proposition 2. *For every states $S = (\Phi,\Psi)$ and S' such that $S \stackrel{\cdot}{\Longrightarrow}^n S'$ using only **B** rules, n is polynomially bounded in the size of* $\mathrm{im}(\Phi)$.

This is due to the fact that frames are one-to-one and that the rule **B.2** only adds deduction facts $M \rhd t$ such that t is a subterm of an existing term in Φ. Hence, for proving termination, we observe that it is sufficient to provide a function s mapping each frame Φ to a finite set of terms $s(\Phi)$ including the subterms of $\mathrm{im}(\Phi)$ and such that rule **A.2** only adds deduction facts $M \rhd t$ satisfying $t \in s(\Phi)$.

For subterm theories, we obtain polynomial termination by choosing $s(\Phi)$ to be the subterms of $\mathrm{im}(\Phi)$ together with the ground right-hand sides of $\mathcal{R}$.

Proposition 3. *Let* E *be a convergent subterm theory. For every* $S = (\Phi, \Psi)$ *and* S' *such that* $S \stackrel{\cdot}{\Longrightarrow}^n S'$, n *is polynomially bounded in the size of* $\mathrm{im}(\Phi)$.

To conclude that deduction and static equivalence are decidable in polynomial time [2], we need to show that the deduction facts and the equations are of polynomial size. This requires a DAG representation for terms and visible equations. For our implementation, we have chosen not to use DAGs for the sake of simplicity (and perhaps efficiency) since DAGs require much heavier data structures. However, similar techniques as those described in [2] would apply to implement our procedure using DAGs.

For proving termination of the prefix theory, we let $s(\Phi)$ be the minimal set containing Φ, closed by subterm and such that $\mathsf{enc}(t_1, k) \in s(\Phi)$ whenever $\mathsf{enc}(\langle t_1, t_2\rangle, k) \in s(\Phi)$. We then deduce that deduction and static equivalence are decidable for the equational theory $\mathsf{E}_{\mathsf{pref}}$, which is a new decidability result.

A criterion to ensure termination. We now provide a semantic criterion that more generally explains why our procedure succeeds on theories previously known to be decidable [2]. This criterion intuitively states that the set of deducible terms from any initial frame φ should be equivalent to a set of *syntactically* deducible terms. Provided that failures are prevented and assuming a *fair* strategy for rule application, we prove that this criterion is a necessary and sufficient condition for our procedure to terminate.

Definition 5 (fair derivation). *An infinite derivation* $(\Phi_0, \Psi_0) \Longrightarrow \ldots \Longrightarrow (\Phi_n, \Psi_n) \Longrightarrow \ldots$ *is* fair *iff along this derivation,*

(a) ***B*** *rules are applied with greatest priority, and*
(b) whenever a ***A*** *rule is applicable for some instance* $(l \to r, D, t_1, \ldots, t_n, \ldots)$, *eventually the same instance of rule is applied during the derivation.*

Fairness implies that any deducible term is eventually syntactically deducible.

Lemma 4. *Let* $S_0 = (\Phi_0, \Psi_0) \Longrightarrow \ldots \Longrightarrow (\Phi_n, \Psi_n) \Longrightarrow \ldots$ *be an infinite fair derivation from a state* S_0. *For every ground term* t *such that* $\Phi_0 \vdash_{\mathsf{E}} t$, *either* $(\Phi_0, \Psi_0) \Longrightarrow^* \bot$ *or there exists* i *such that* $\Phi_i \vdash t{\downarrow_{\mathcal{R}}}$.

Proposition 4 (criterion for saturation). *Let* φ *be an initial frame such that* $\mathrm{Init}(\varphi) \not\Longrightarrow^* \bot$. *The following conditions are equivalent:*

(i) There exists a saturated couple (Φ, Ψ) *such that* $\mathrm{Init}(\varphi) \Longrightarrow^* (\Phi, \Psi)$.
(ii) There exists a (finite) initial frame φ_s *such that for every term* t, t *is deducible from* φ *modulo* E *iff* $t{\downarrow_{\mathcal{R}}}$ *is syntactically deducible from* φ_s.
(iii) There exists no fair infinite derivation starting from $\mathrm{Init}(\varphi)$.

Together with the syntactic criterion described in Section 6.1, this criterion (Property *(ii)*) allows us to prove decidability of deduction and static equivalence for layered convergent theories that belong to the class of *locally stable*

theories defined in [2]. As a consequence, our procedure always saturates for the theories of blind signatures and homomorphic encryption since those theories are layered and have been proved locally stable [2]. Other examples of layered convergent theories enjoying this criterion can be found in [2] (e.g. a theory of addition).

7 Implementation: The YAPA Tool

YAPA is an Ocaml implementation[1] of the saturation procedure presented in Section 4, using by default the optimized function Ctx defined in Section 6, and a fair strategy of rule application (see Definition 5).

The tool takes as input an equational theory described by a finite convergent rewrite system, as well as frame definitions and queries. A few optimizations may be activated for subterm theories, e.g. to accelerate normalization. The procedure starts by computing the decompositions of the rewrite system. Provided that the rewrite rules are given in an order compatible with the sets $\mathcal{R}_0 \subseteq \ldots \subseteq \mathcal{R}_{N+1}$ of Definition 4, it is able to recognize (fully or partially) layered theories and to pre-compute the associated contexts C related to condition (ii) of this definition, and exploited by the function Ctx in use for eliminating failure cases.

We have conducted several experiments on a PC Intel Core 2 Duo at 2.4 GHz with 2 Go RAM for various equational theories (see below) and found that YAPA provides an efficient way to check static equivalence and deducibility.

Equational theory	$\mathsf{E}_{\mathsf{enc}}$ $n=10$	$\mathsf{E}_{\mathsf{enc}}$ $n=14$	$\mathsf{E}_{\mathsf{enc}}$ $n=16$	$\mathsf{E}_{\mathsf{enc}}$ $n=18$	$\mathsf{E}_{\mathsf{enc}}$ $n=20$	$\mathsf{E}_{\mathsf{blind}}$	$\mathsf{E}_{\mathsf{pref}}$	$\mathsf{E}_{\mathsf{hom}}$	$\mathsf{E}_{\mathsf{add}}$
Execution time	$<$ 1s	1,7s	8s	30s	$<$ 3min	$<$ 1s	$<$ 1s	$<$ 1s	$<$ 1s

For the case of $\mathsf{E}_{\mathsf{enc}}$, we have run YAPA on the frames $\varphi_n = \{\mathsf{w}_1 \rhd t_n^0, \mathsf{w}_2 \rhd \mathsf{c}_0, \mathsf{w}_3 \rhd \mathsf{c}_1\}$ and $\varphi'_n = \{\mathsf{w}_1 \rhd t_n^1, \mathsf{w}_2 \rhd \mathsf{c}_0, \mathsf{w}_3 \rhd \mathsf{c}_1\}$, where $t_0^i = \mathsf{c}_i$ and $t_{n+1}^i = \langle \mathsf{enc}(t_n^i, \mathsf{k}_n^i), \mathsf{k}_n^i \rangle$, $i \in \{0, 1\}$. These examples allow us to increase the (tree, non-DAG) size of the distinguishing tests exponentially, while the sizes of the frames grow linearly. Despite the size of the output, we have observed satisfactory performances for the tool. We have also experimented YAPA on several convergent theories, e.g. $\mathsf{E}_{\mathsf{blind}}$, $\mathsf{E}_{\mathsf{hom}}$, $\mathsf{E}_{\mathsf{pref}}$ and the theory of addition $\mathsf{E}_{\mathsf{add}}$ defined in [2].

In comparison with the tool ProVerif [9,10], here instrumented to check static equivalences, our test samples suggest a running time between one and two orders of magnitude faster for YAPA. Also we did not succeed in making ProVerif terminate on the two theories $\mathsf{E}_{\mathsf{hom}}$ and $\mathsf{E}_{\mathsf{add}}$. Of course, these results are not entirely surprising given that ProVerif is tailored for the more general (and difficult) problem of protocol (in)security under active adversaries. In particular ProVerif's initial preprocessing of the rewrite system appears more substantial than ours and does not terminate on the theories $\mathsf{E}_{\mathsf{hom}}$ and $\mathsf{E}_{\mathsf{add}}$ (although termination is guaranteed for linear or subterm-convergent theories [10]).

[1] Freely available at `http://www.lsv.ens-cachan.fr/~baudet/yapa/`

Altogether, these results suggest that YAPA significantly improves the state of the art for checking deducibility and static equivalence under convergent theories, both from practical and theoretical perspectives.

References

1. Abadi, M., Baudet, M., Warinschi, B.: Guessing attacks and the computational soundness of static equivalence. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 398–412. Springer, Heidelberg (2006)
2. Abadi, M., Cortier, V.: Deciding knowledge in security protocols under equational theories. Theoretical Computer Science 387(1-2), 2–32 (2006)
3. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: 28th ACM Symposium on Principles of Programming Languages (POPL 2001), pp. 104–115. ACM Press, New York (2001)
4. Anantharaman, S., Narendran, P., Rusinowitch, M.: Intruders with caps. In: Baader, F. (ed.) RTA 2007. LNCS, vol. 4533, pp. 20–35. Springer, Heidelberg (2007)
5. Baudet, M.: Deciding security of protocols against off-line guessing attacks. In: 12th ACM Conference on Computer and Communications Security (CCS 2005), pp. 16–25. ACM Press, New York (2005)
6. Baudet, M.: Sécurité des protocoles cryptographiques : aspects logiques et calculatoires. Thèse de doctorat, LSV, ENS Cachan, France (2007)
7. Baudet, M., Cortier, V., Delaune, S.: YAPA: A generic tool for computing intruder knowledge. Research Report LSV-09-03, Laboratoire Spécification et Vérification, ENS Cachan, France, 28 pages (Feburary 2009)
8. Baudet, M., Cortier, V., Kremer, S.: Computationally sound implementations of equational theories against passive adversaries. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 652–663. Springer, Heidelberg (2005)
9. Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In: 14th Computer Security Foundations Workshop (CSFW 2001), pp. 82–96. IEEE Comp. Soc. Press, Los Alamitos (2001)
10. Blanchet, B., Abadi, M., Fournet, C.: Automated verification of selected equivalences for security protocols. Journal of Logic and Algebraic Programming 75(1), 3–51 (2008)
11. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: An NP decision procedure for protocol insecurity with XOR. In: 18th IEEE Symposium on Logic in Computer Science (LICS 2003). IEEE Comp. Soc. Press, Los Alamitos (2003)
12. Comon-Lundh, H., Shmatikov, V.: Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In: 18th IEEE Symposium on Logic in Computer Science (LICS 2003). IEEE Computer Society Press, Los Alamitos (2003)
13. Corin, R., Doumen, J., Etalle, S.: Analysing password protocol security against off-line dictionary attacks. In: 2nd International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004). ENTCS (2004)
14. Cortier, V., Delaune, S.: Deciding knowledge in security protocols for monoidal equational theories. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790, pp. 196–210. Springer, Heidelberg (2007)
15. Cortier, V., Delaune, S., Lafourcade, P.: A survey of algebraic properties used in cryptographic protocols. Journal of Computer Security 14(1), 1–43 (2006)

16. Delaune, S., Jacquemard, F.: A decision procedure for the verification of security protocols with explicit destructors. In: 11th ACM Conference on Computer and Communications Security (CCS 2004), pp. 278–287 (2004)
17. Delaune, S., Kremer, S., Ryan, M.D.: Verifying privacy-type properties of electronic voting protocols. Journal of Computer Security (to appear) (2008)
18. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996)
19. Millen, J., Shmatikov, V.: Constraint solving for bounded-process cryptographic protocol analysis. In: 8th ACM Conference on Computer and Communications Security (CCS 2001) (2001)

Well-Definedness of Streams by Termination

Hans Zantema[1,2]

[1] Department of Computer Science, TU Eindhoven, P.O. Box 513,
5600 MB Eindhoven, The Netherlands
H.Zantema@tue.nl
[2] Institute for Computing and Information Sciences, Radboud University
Nijmegen, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands

Abstract. Streams are infinite sequences over a given data type. A stream specification is a set of equations intended to define a stream. We propose a transformation from such a stream specification to a TRS in such a way that termination of the resulting TRS implies that the stream specification admits a unique solution. As a consequence, proving such well-definedness of several interesting stream specifications can be done fully automatically using present powerful tools for proving TRS termination.

1 Introduction

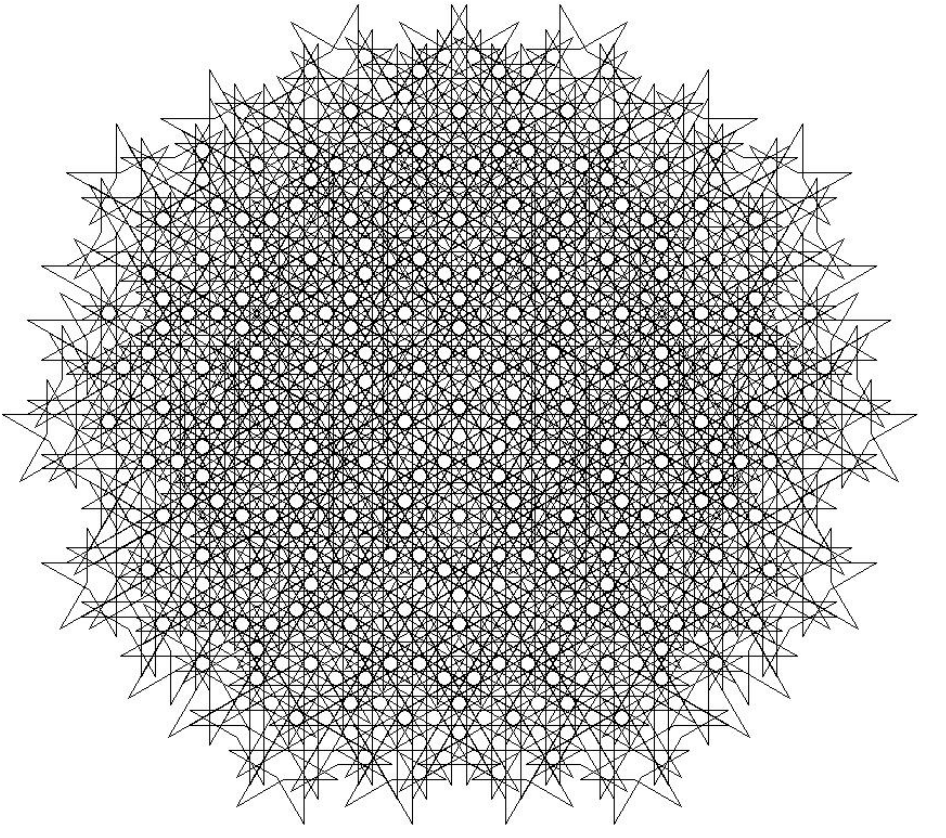
Streams are among the simplest data types in which the objects are infinite. We consider streams to be maps from the natural numbers to some data type D. The basic constructor for streams is the operator ':' mapping a data element d and a stream s to a new stream $d : s$ by putting d in front of s. Using this operator we can define streams by equations. For instance, the stream zeros only consisting of 0's can be defined by the single equation $\mathsf{zeros} = 0 : \mathsf{zeros}$. More complicated streams are defined using stream functions. For instance, the boolean *Fibonacci stream* Fib is defined[1] to be the fixpoint of the function f defined by

$$f(0 : \sigma) = 0 : 1 : f(\sigma), \quad f(1 : \sigma) = 0 : f(\sigma).$$

It turns out that $\mathsf{Fib} = 0 : c$ for the stream c defined by $c = 1 : f(c)$. Although these stream definitions are extremely simple, the resulting streams are typically non-periodic and have remarkable properties. For instance, one can make a *turtle visualization* (see also `http://www.win.tue.nl/~hzantema/str.html`) as follows. Choose an initial drawing direction and traverse the elements of the stream Fib as follows: if the symbol 0 is read then the drawing direction is moved 30 degrees to the right; if the symbol 1 is read then the drawing direction is moved 150 degrees to the left. In both cases after doing so a line of unit length is drawn. Then after 200.000 steps the following picture is obtained.

[1] In [1] it is called infinite Fibonacci word. It can also be defined as the limit of the strings ϕ_i where $\phi_1 = 1$, $\phi_2 = 0$, $\phi_{i+2} = \phi_{i+1}\phi_i$ for $i \geq 1$, showing the relationship with Fibonacci numbers.

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 164–178, 2009.
© Springer-Verlag Berlin Heidelberg 2009



Streams have been studied extensively, e.g. in [1]. In this paper we consider stream specifications consisting of a set of equations like above we did for the Fibonacci stream. We address the most fundamental question one can think of: does such a set of equations admits a unique solution as constants and functions on streams? This is not always the case. For instance, every f mapping $x : \sigma$ to $x : c$ for any constant stream c satisfies the stream specification

$$f(x : \sigma) = x : g(f(\sigma)), \quad g(x : \sigma) = \sigma,$$

where g is the tail function removing the first element of the stream.

Intuitively this notion of well-definedness is closely related to termination of the process of unfolding definitions. The past ten years showed up a remarkable progress in techniques and implementations for proving termination of rewrite systems [2,5,9]. One of the objectives of this paper is to exploit this power for proving well-definedness of stream specifications. In our approach we introduce fresh operators head and tail intended to observe streams. We present a transformation of the specification to its *observational variant*. This is a TRS mimicking the stream specification in such a way that head or tail applied on any stream constant or stream function can always be rewritten. This transformation is

straightforward and easy to implement; an implementation for boolean stream specifications, both in Windows an Linux, together with several examples, is found in `http://www.win.tue.nl/~hzantema/str.zip`.

The main result of this paper states that if the observational variant of a specification is terminating, then the specification admits a unique solution. It turns out that for several interesting cases termination of the observational variant of a specification can be proved by termination tools like AProVE [4] or TTT2 [7]. This provides a new technique to prove well-definedness of stream specifications fully automatically, applying for cases where earlier approaches fail. Our main result appears in two variants:

- a variant restricting to ground terms for general stream specifications (Theorem 1), and
- a variant generalizing to all streams for stream specifications not depending on particular data elements (Theorem 2).

By an example we show that the approach does not work for general stream specifications and functions applied on all streams. Moreover, we show that our technique is not complete: the fixpoint definition of the Fibonacci stream as we just gave is a well-defined stream specification for which the observational variant is non-terminating.

Proving well-definedness in stream specification is closely related to proving equality of streams. A standard approach for this is co-induction [11]: two streams or stream functions are equal if a bisimulation can be found between them. Finding such an arbitrary bisimulation is a hard problem in the general setting, but restricting to circular co-induction [6] finding this automatically is tractable. A strong tool doing so is Circ [8]. The tool Circ focuses on proving equality, but proving well-definedness of a function f can also be proved by equality as long as the equations for f are orthogonal: take a copy f' of f with the same equations, and prove $f = f'$. For many examples this works well, but there are also small stream specifications for which our approach succeeds in proving well-definedness and Circ fails. Conversely our approach can be used to prove equality of two streams: if one stream satisfies the specification of the other one, and this specification is well-defined, then the streams are equal. The input format of Circ differs from what we call stream specifications: head and tail are already building blocks and the Circ input is essentially the same as what we call the observational variant.

Another closely related topic is *productivity* of stream specifications, as studied by [3]. Productive stream specifications are always well-defined. Conversely we will give an example (Example 4) of a stream specification that is well-defined, but not productive. Our format of stream specifications is strongly inspired by [3]. In [3] a technique is developed for establishing productivity fully automatically for a restricted class of stream specifications. In particular, only a very mild type of nesting in the right hand sides of the rule is allowed. Our technique typically applies where these restrictions do not hold.

Both stream equality [10] and productivity [12] have been proved to be Π^0_2-complete, hence undecidable. By similar Turing machine construction the same is expected to hold for stream well-definedness.

This paper is structured as follows. In Section 2 we present the basics of stream specifications and their models. In Section 3 we define the transformation of a stream specification to its observational variant. In Section 4 we present and prove the main theorem: if the observational variant is terminating then restricted to ground terms the specification has a unique model. In Section 5 we show that this restriction to ground terms may be removed in case the stream specification is data independent: left hand sides of rules do not contain data values. In Section 6 we discuss fixpoints and prove incompleteness. We conclude in Section 7.

2 Streams: Specifications and Models

In stream specifications we have two sorts: s (stream) and d (data). We assume the set D of data elements to consist of the unique normal forms of ground terms over some signature Σ_d with respect to some terminating orthogonal rewrite system R_d over Σ_d. Here all symbols of Σ_d are of type $d^n \to d$ for some $n \geq 0$. We assume a particular symbol : having type $d \times s \to s$. For giving the actual stream specification we need a set Σ_s of stream symbols, each being of type $d^n \times s^m \to s$ for $n, m \geq 0$. Now terms of sort s are defined inductively as follows:

- a variable of sort s is a term of sort s,
- if $f \in \Sigma_s$ is of type $d^n \times s^m \to s$, $u_1, \ldots, u_n$ are terms over Σ_d and $t_1, \ldots, t_m$ are terms of sort s, then $f(u_1, \ldots, u_n, t_1, \ldots, t_m)$ is a term of sort s,
- if u is a term over Σ_d and t is a term of sort s, then $u : t$ is a term of sort s.

As a notational convention variables of sort d will be denoted by x, y, terms of sort d by u, u_i, variables of sort s by σ, τ, and terms of sort s by t, t_i.

Definition 1. *A* stream specification $(\Sigma_d, \Sigma_s, R_d, R_s)$ *consists of* Σ_d, Σ_s, R_d *as given before, and a set* R_s *of rewrite rules of the shape*

$$f(u_1, \ldots, u_n, t_1, \ldots, t_m) \to t,$$

where

- $f \in \Sigma_s$ *is of type* $d^n \times s^m \to s$,
- *for every* $i = 1, \ldots, n$ *the term* u_i *is either a variable of sort* d *or* $u_i \in D$,
- *for every* $i = 1, \ldots, m$ *the term* t_i *is either a variable of sort* s, *or* $t_i = x : \sigma$ *where* x *is a variable of sort* d *and* σ *is a variable of sort* s,
- t *is any term of sort* s,
- *every ground term of sort* s *being in normal form with respect to* R_d *matches with the left hand side of exactly one rule from* R_s.

Due to these requirements $R_s \cup R_d$ is orthogonal. Sometimes we call R_s a stream specification: in that case Σ_d, Σ_s consist of the symbols of sort d, s, respectively, occurring in R_s, and $R_d = \emptyset$. Rules $\ell \to r$ in R_s are often written as $\ell = r$.

Example 1. For specifying the Thue Morse sequence the data elements are $0, 1$, and a data operation not is used. The data rewrite system R_d consists of the two rules $\mathsf{not}(0) \to 1$ and $\mathsf{not}(1) \to 0$. The rewrite system R_s consists of the rules

$$\begin{array}{ll} \mathsf{morse} \to 0 : \mathsf{zip}(\mathsf{inv}(\mathsf{morse}), \mathsf{tail}(\mathsf{morse})) & \mathsf{tail}(x : \sigma) \to \sigma \\ \mathsf{inv}(x : \sigma) \to \mathsf{not}(x) : \mathsf{inv}(\sigma) & \mathsf{zip}(x : \sigma, \tau) \to x : \mathsf{zip}(\tau, \sigma) \end{array}$$

Definition 1 is closely related to the definition of stream specification in [3]. In fact there are two differences:

- We want to specify streams for every ground term of sort s, while in [3] there is a designated constant to be specified.
- The restriction on left hand sides is stronger than the exhaustiveness from [3]. However, by introducing fresh symbols and rules for defining these fresh symbols, every stream specification in the format of [3] can be unfolded to a stream specification in our format.

For instance, the function f in the introduction to define the Fibonacci stream does not meet our requirements since the argument $0 : \sigma$ in the left hand side $f(0 : \sigma)$ is not of the right shape. Introducing a fresh symbol g and unfolding yields

$$\begin{array}{ll} f(x : \sigma) = g(x, \sigma) & g(0, \sigma) = 0 : 1 : f(\sigma) \\ & g(1, \sigma) = 0 : f(\sigma) \end{array}$$

satisfying our format.

Stream specifications are intended to specify streams for the constants in Σ_s, and stream functions for the other elements of Σ_s. The combination of these streams and stream functions is what we will call a *stream model*.

More precisely, a *stream* over D is a map from the natural numbers to D. Write D^ω for the set of all streams over D. In case of $D = \emptyset$ we have $D^\omega = \emptyset$; in case of $\#D = 1$ we have $\#D^\omega = 1$. So in non-degenerate cases we have $\#D \geq 2$.

It seems natural to require that stream functions in a stream model are defined on all streams. However, it turns out that several desired properties do not hold when requiring this. Therefore we allow stream functions to be defined on some set $S \subseteq D^\omega$ for which every ground term can be interpreted in S.

Definition 2. *A* stream model *is defined to consist of a set $S \subseteq D^\omega$ and a set of functions $[f]$ for every $f \in \Sigma_s$, where $[f] : D^n \times S^m \to S$ if the type of f is $d^n \times s^m \to s$.*

For a ground term u over Σ_d write $\mathbf{NF}(u)$ for its R_d-normal form. For $f \in \Sigma_d$ and $u_1, \ldots, u_n \in D$ we define $[f(u_1, \ldots, u_n)] = \mathbf{NF}(f(u_1, \ldots, u_n))$. We write $\mathcal{T}_s$

for the set of ground terms of sort s. For $t \in \mathcal{T}_s$ the stream interpretation $[t]$ in the stream model $(S, ([f])_{f \in \Sigma_s})$ is defined inductively by:

$$\begin{aligned} [f(u_1, \ldots, u_n, t_1, \ldots, t_m)] &= [f]([u_1], \ldots, [u_n], [t_1], \ldots, [t_m]) \quad \text{for } f \in \Sigma_s \\ [u : t](0) &= [u] \\ [u : t](i) &= [t](i-1) \qquad\qquad\qquad\qquad \text{for } i > 0 \end{aligned}$$

for all ground terms u, u_i of sort d and all ground terms t, t_i of sort s.

So in a stream model:

- every data operator is interpreted by its corresponding term constructor, after which the result is reduced to normal form,
- every stream operator f is interpreted by the given function $[f]$, and
- the operator : applied on a data element d and a stream s is interpreted by putting d on the first position and shifting every stream element of s to its next position.

Definition 3. *A stream model $(S, ([f])_{f \in \Sigma_s})$ is said to* satisfy *a stream specification $(\Sigma_d, \Sigma_s, R_d, R_s)$ if $[\ell\rho] = [r\rho]$ for every rule $\ell \to r$ in R_s and every ground substitution ρ. We also say that the specification* admits *the model.*

Now we can express the desired well-definedness of a stream specification more precisely: there is exactly one stream model $(S, ([f])_{f \in \Sigma_s})$ satisfying the stream specification for which $S = \{[t] \mid t \in \mathcal{T}_s\}$. This is not always the case: if $\#D > 1$ and R_s consists of the rule $c \to c$ there is not a unique $[c]$ since every stream satisfies the specification. Less trivial is the boolean stream specification

$$c = 0 : f(c), \qquad f(x : \sigma) = \sigma,$$

in which $[f]$ can be chosen to be the tail function and $[c]$ be any stream starting with 0, showing non-uniqueness of stream models.

3 The Observational Variant

In this paper we define a transformation Obs transforming the original TRS R_s to its *observational variant* $\mathsf{Obs}(R_s)$. The basic idea is that the streams are observed by two auxiliary operator head and tail, of which head picks the first element of the stream and tail removes the first element from the stream, and that for every $t \in \mathcal{T}_s$ of type stream both $\mathsf{head}(t)$ and $\mathsf{tail}(t)$ can be rewritten by $\mathsf{Obs}(R_s)$.

The main result of this paper is that if $\mathsf{Obs}(R_s) \cup R_d$ is terminating for a given specification $(\Sigma_d, \Sigma_s, R_d, R_s)$, then it admits a unique model $(S, ([f])_{f \in \Sigma_s})$ satisfying $S = \{[t] \mid t \in \mathcal{T}_s\}$. As a consequence, the specification uniquely defines a corresponding stream $[t]$ for every $t \in \mathcal{T}_s$.

We define $\mathsf{Obs}(R_s)$ in two steps. First we define $\mathsf{P}(R_s)$ obtained from R_s by modifying the rules as follows. By definition every rule of R_s is of the shape

$$f(u_1, \ldots, u_n, t_1, \ldots, t_m) \to t$$

where for every $i = 1, \ldots, m$ the term t_i is either a variable of sort s, or $t_i = x : \sigma$ where x is a variable of sort d and σ is a variable of sort s. In case $t_i = x : \sigma$ then in the left hand side of the rule the subterm t_i is replaced by σ, while in the right hand side of the rule every occurrence of x is replaced by $\mathsf{head}(\sigma)$ and every occurrence of σ is replaced by $\mathsf{tail}(\sigma)$.

For example, the zip rule in Example 1 will be replaced by

$$\mathsf{zip}(\sigma, \tau) \to \mathsf{head}(\sigma) : \mathsf{zip}(\tau, \mathsf{tail}(\sigma)).$$

Now we are ready to define Obs.

Definition 4. *Let $(\Sigma_d, \Sigma_s, R_d, R_s)$ be a stream specification. Let $\mathsf{P}(R_s)$ be defined as above. Then $\mathsf{Obs}(R_s)$ is the TRS over $\Sigma_d \cup \Sigma_s \cup \{:, \mathsf{head}, \mathsf{tail}\}$ consisting of*

- *the two rules*
$$\mathsf{head}(x : \sigma) \to x, \quad \mathsf{tail}(x : \sigma) \to \sigma,$$
- *for every rule in $\mathsf{P}(R_s)$ of the shape $\ell \to u : t$ the two rules*
$$\mathsf{head}(\ell) \to u, \quad \mathsf{tail}(\ell) \to t,$$
- *for every rule in $\mathsf{P}(R_s)$ of the shape $\ell \to r$ with $\mathsf{root}(r) \neq :$ the two rules*
$$\mathsf{head}(\ell) \to \mathsf{head}(r), \quad \mathsf{tail}(\ell) \to \mathsf{tail}(r).$$

Example 2. For the TRS R_s given in Example 1 we rename the symbol tail by tail0 in order to keep the symbol tail for the fresh symbol introduced in the Obs construction. Then the TRS $\mathsf{Obs}(R_s)$ consists of the following rules:

$$\begin{aligned}
\mathsf{head}(x : \sigma) &\to x \\
\mathsf{tail}(x : \sigma) &\to \sigma \\
\mathsf{head}(\mathsf{morse}) &\to 0 \\
\mathsf{tail}(\mathsf{morse}) &\to \mathsf{zip}(\mathsf{inv}(\mathsf{morse}), \mathsf{tail}(\mathsf{morse})) \\
\mathsf{head}(\mathsf{inv}(\sigma)) &\to \mathsf{not}(\mathsf{head}(\sigma)) \\
\mathsf{tail}(\mathsf{inv}(\sigma)) &\to \mathsf{inv}(\mathsf{tail}(\sigma))
\end{aligned}
\qquad
\begin{aligned}
\mathsf{head}(\mathsf{tail0}(\sigma)) &\to \mathsf{head}(\mathsf{tail}(\sigma)) \\
\mathsf{tail}(\mathsf{tail0}(\sigma)) &\to \mathsf{tail}(\mathsf{tail}(\sigma)) \\
\mathsf{head}(\mathsf{zip}(\sigma, \tau)) &\to \mathsf{head}(\sigma) \\
\mathsf{tail}(\mathsf{zip}(\sigma, \tau)) &\to \mathsf{zip}(\tau, \mathsf{tail}(\sigma))
\end{aligned}$$

Together with the rules $\mathsf{not}(0) \to 1$ and $\mathsf{not}(1) \to 0$ from R_d this TRS is terminating as can easily be proved fully automatically by AProVE [4] or TTT2 [7]. As a consequence, the result of this paper states that the specification uniquely defines a stream for every ground term of type s, in particular for morse.

4 The Main Theorem

We start this section by presenting our main theorem.

Theorem 1. *Let $(\Sigma_d, \Sigma_s, R_d, R_s)$ be a stream specification for which the TRS $\mathsf{Obs}(R_s) \cup R_d$ is terminating. Then the stream specification admits a unique model $(S, ([f])_{f \in \Sigma_s})$ satisfying $S = \{[t] \mid t \in \mathcal{T}_s\}$.*

Before proving the theorem we show by an example why it is essential to restrict to $S = \{[t] \mid t \in \mathcal{T}_s\}$ rather than choosing $S = D^\omega$. A degenerate example is obtained if there are no constants of sort s, and hence $\mathcal{T}_s = \emptyset$. More interesting is the following.

Example 3. Let the boolean stream specification consist of $R_d = \emptyset$ and R_s consisting of the following rules:

$$\begin{array}{ll} c \to 1 : c & g(0,\sigma) \to f(\sigma) \\ f(x:\sigma) \to g(x,\sigma) & g(1,\sigma) \to 1 : f(\sigma) \end{array}$$

So f tries to remove all 0's from its argument. For streams containing infinitely many 0's this may be problematic. Note that by the symbols c, :, 0 and 1 only the streams with finitely many 0's can be constructed, for ground terms this problem does not arise. Indeed the TRS $\mathsf{Obs}(R_s) \cup R_d$ is terminating, and by Theorem 1 the specification admits a unique model $(S, ([f])_{f\in\Sigma_s})$ satisfying $S = \{[t] \mid t \in \mathcal{T}_s\}$. However, when extending to all streams the function $[f] : D^\omega \to D^\omega$ is not uniquely defined, even if we strengthen the requirement of $[\ell\rho] = [r\rho]$ for all rules $\ell \to r$ and all ground substitutions ρ to an open variant in which the σ's in the rules are replaced by arbitrary streams. Write ones and zeros for the streams only consisting of ones, resp. zeros. Two distinct models $[\cdot]_1$ and $[\cdot]_2$ satisfying the stream specification are defined by:

$$[c]_1 = [f]_1(s) = [g]_1(u,s) = \mathsf{ones} \text{ for all } s \in D^\omega, u \in D,$$

and $[c]_2 = [f]_2(s) = [g]_2(u,s) = \mathsf{ones}$ for $u \in D$ and streams s containing infinitely many ones, and $[f]_2(s) = 1^n : \mathsf{zeros}$, $[g]_2(u,s) = [f]_2(u:s)$ for $u \in D$ and streams s containing $n < \infty$ ones.

Now we arrive at the proof of Theorem 1. The plan of the proof is as follows.

- First we construct a function $[\cdot]_1 : \mathcal{T}_s \to D^\omega$, and choose $S_1 = \{[t]_1 \mid t \in \mathcal{T}_s\}$.
- Next we show that if $[t_i]_1 = [t'_i]_1$ for $i = 1, \ldots, m$, then

$$[f(u_1, \ldots, u_n, t_1, \ldots, t_m)]_1 = [f(u_1, \ldots, u_n, t'_1, \ldots, t'_m)]_1,$$

  by which $[f]_1$ is well-defined and we have a model $(S_1, ([f]_1)_{f\in\Sigma_s})$.
- We show this model satisfies the specification.
- We show no other model $(S, ([f])_{f\in\Sigma_s})$ satisfies the specification and $S = \{[t] \mid t \in \mathcal{T}_s\}$.

First we define $[t]_1 \in D^\omega$ for any $t \in \mathcal{T}_s$. Since elements of D^ω are functions from $\mathbf{N}$ to D, a function $[t]_1 \in D^\omega$ is defined by defining $[t]_1(n)$ for every $n \in \mathbf{N}$. Due to the assumption of the theorem the TRS $\mathsf{Obs}(R_s) \cup R_d$ is terminating. According to the definition of stream specification the TRS $R_s \cup R_d$ is orthogonal, and by the construction Obs the TRS $\mathsf{Obs}(R_s) \cup R_d$ is orthogonal too. So every ground term of sort d has a unique normal form with respect to $\mathsf{Obs}(R_s) \cup R_d$.

Assume such a normal form contains a symbol from $\Sigma_s \cup \{:\}$. Choose such a symbol with minimal position, that is, closest to the root. Since the term is

of sort d, this symbol is not the root. Hence it has a parent. Due to minimality of position, this parent is either head or tail. Due to the shape of the rules of $\mathsf{Obs}(R_s)$, a rule of $\mathsf{Obs}(R_s)$ is applicable on this parent position, contradicting the normal form assumption. So the normal form only contains symbols from Σ_d. Since it is also a normal form with respect to R_d, such a normal form is an element of D. Now for $t \in \mathcal{T}_s$ and $n \in \mathbf{N}$ we define

$$[t]_1(n) = \text{the normal form of } \mathsf{head}(\mathsf{tail}^n(t)) \text{ with respect to } \mathsf{Obs}(R_s) \cup R_d,$$

in this way defining $[t]_1 \in D^\omega$.

Lemma 1. *Let $\mathsf{Obs}(R_s) \cup R_d$ be terminating. Let $f \in \Sigma_s$ of type $d^n \times s^m \to s$. Let $u_1, \ldots, u_n \in D$ and $t_1, \ldots, t_m, t'_1, \ldots, t'_m \in \mathcal{T}_s$ satisfying $[t_i]_1 = [t'_i]_1$ for $i = 1, \ldots, m$. Then*

$$[f(u_1, \ldots, u_n, t_1, \ldots, t_m)]_1 = [f(u_1, \ldots, u_n, t'_1, \ldots, t'_m)]_1.$$

Proof. First we extend the definition of $[\cdot]_1$ to all ground terms over $\Sigma_s \cup \Sigma_d \cup \{:, \mathsf{head}, \mathsf{tail}\}$. For ground terms t of sort s we define it by $[t]_1(n) =$ the normal form of $\mathsf{head}(\mathsf{tail}^n(t))$ with respect to $\mathsf{Obs}(R_s) \cup R_d$, and for ground terms u of sort d we define $[u]_1$ to be the normal form of u with respect to $\mathsf{Obs}(R_s) \cup R_d$. We prove the following claim.

> **Claim 1:** Let $[t]_1 = [t']_1$ for $t, t' \in \mathcal{T}_s$. Let T be a ground term over $\Sigma_s \cup \Sigma_d \cup \{:, \mathsf{head}, \mathsf{tail}\}$ of sort s containing t as a subterm. Let T' be obtained from T by replacing zero or more occurrences of the subterm t by t'. Then
>
> $$[\mathsf{head}(T)]_1 = [\mathsf{head}(T')]_1.$$

Let $>$ be the well-founded order on ground terms being the strict part of $\geq$ defined by

$$v \geq v' \iff v' \text{ is a subterm } v'' \text{ such that } v \to^*_{\mathsf{Obs}(R_s) \cup R_d} v''.$$

We prove the claim for every such term $\mathsf{head}(T)$ by noetherian induction on $>$.

Claim 1 is trivial if $t = T$, so we may assume that $T = f(u_1, \ldots, u_n, t_1, \ldots, t_m)$ such that t occurs in $u_1, \ldots, u_n, t_1, \ldots, t_m$, and either $f \in \Sigma_s \cup \{:, \mathsf{tail}\}$, and $T' = f(u'_1, \ldots, u'_n, t'_1, \ldots, t'_m)$. For every subterm of u_i of the shape $\mathsf{head}(\cdots)$ we may apply the induction hypothesis, yielding $[u_i]_1 = [u'_i]_1 = d_i$ for all i, defining $d_i \in D$.

In case the root of T is not tail we rewrite

$$\mathsf{head}(T) \to^*_{\mathsf{Obs}(R_s) \cup R_d} \mathsf{head}(f(d_1, \ldots, d_n, t_1, \ldots, t_m)$$

by the rule $\mathsf{head}(f(\cdots)) \to \cdots$ in $\mathsf{Obs}(R_s)$, yielding a term U of sort d. The only way such a term can contain t as a subterm is by $U = C[\mathsf{head}(V_1), \ldots, \mathsf{head}(V_k)]$ where t is a subterm of some of the V_i and C is composed from Σ_d. By the induction hypothesis we obtain $[\mathsf{head}(V_i)]_1 = [\mathsf{head}(V'_i)]_1$ for V'_i obtained from V_i by

replacing zero ore more occurrences of t by t'. Hence $[\mathsf{head}(T)]_1 = C[\mathsf{head}(V_1), \ldots, \mathsf{head}(V_k)]]_1 = C[\mathsf{head}(V_1'), \ldots, \mathsf{head}(V_k')]]_1 = [\mathsf{head}(T')]_1$.

In case the root of T is tail then write $T = \mathsf{tail}^i(f(\cdots)) \to^*_{\mathsf{Obs}(R_s) \cup R_d} \mathsf{tail}^i(f(d_1, \ldots, d_n, t_1, \ldots, t_m)$ for $f \in \Sigma_s \cup \{:\}$. This can be rewritten by the rule $\mathsf{tail}(f(\cdots)) \to \cdots$ in $\mathsf{Obs}(R_s)$, yielding V. On the same position using the same rule we can rewrite $T' \to_{\mathsf{Obs}(R_s)} V'$ for V' obtained from V by replacing one ore more occurrences of t by t'. Applying the induction hypothesis gives $[\mathsf{head}(V)]_1 = [\mathsf{head}(V')]_1$ yielding

$$[\mathsf{head}(T)]_1 = [\mathsf{head}(V)]_1 = [\mathsf{head}(V')]_1 = [\mathsf{head}(T')]_1,$$

concluding the proof of Claim 1.

> **Claim 2:** Let $[t]_1 = [t']_1$ for $t, t' \in \mathcal{T}_s$. Let T be a ground term over $\Sigma_s \cup \Sigma_d \cup \{:, \mathsf{head}, \mathsf{tail}\}$ of sort s containing t as a subterm. Let T' be obtained from T by replacing one or more occurrences of the subterm t by t'. Then $[T]_1 = [T']_1$.

Claim 2 easily follows from Claim 1 and the observation

$$[T]_1 = [T']_1 \iff \forall i \in \mathbf{N} : [\mathsf{head}(\mathsf{tail}^i(T)]_1 = [\mathsf{head}(\mathsf{tail}^i(T')]_1.$$

Now the lemma follows by applying Claim 2 and replacing t_i by t_i' successively for $i = 1, \ldots, m$. □

Define $S_1 = \{[t]_1 \mid t \in \mathcal{T}_s\}$. For any $f \in \Sigma_s$ of type $d^n \times s^m \to s$ for $u_1, \ldots, u_n \in D$ and $t_1, \ldots, t_m, t_1', \ldots, t_m' \in \mathcal{T}_s$ we now define $[f]_1 : D^n \times S^m \to S$ by

$$[f]_1(u_1, \ldots, u_n, [t_1], \ldots, [t_m]) = [f(u_1, \ldots, u_n, t_1, \ldots, t_m)]_1;$$

Lemma 1 implies that this is well-defined: the result is independent of the choice of the representants in $[t_i]_1$. So $(S_1, ([f]_1)_{f \in \Sigma_s})$ is a model.

Next we will prove that it satisfies the specification, and essentially is the only one doing so.

Lemma 2. *Let $\ell \to r \in R_s$ and let ρ be a substitution. Then*

- *there is a term t such that $\mathsf{head}(\ell\rho) \to^*_{\mathsf{Obs}(R_s)} t$ and $\mathsf{head}(r\rho) \to^*_{\mathsf{Obs}(R_s)} t$, and*
- *there is a term t such that $\mathsf{tail}(\ell\rho) \to^*_{\mathsf{Obs}(R_s)} t$ and $\mathsf{tail}(r\rho) \to^*_{\mathsf{Obs}(R_s)} t$.*

Proof. Let f be the root of ℓ. Define ρ' by $\sigma\rho' = x\rho : \sigma\rho$ for every argument of the shape $x : \sigma$ of f in ℓ, and ρ' coincides with ρ on all other variables. Then $\mathsf{head}(\ell\rho) = \ell'\rho'$ for some rule in $\ell' \to r'$ in $\mathsf{Obs}(R_s)$. Now a common reduct t of $r'\rho'$ and $\mathsf{head}(r\rho)$ is obtained by applying the rule $\mathsf{head}(x : \sigma) \to x$ zero or more times. This yields $\mathsf{head}(\ell\rho) = \ell'\rho' \to_{\mathsf{Obs}(R_s)} r'\rho' \to^*_{\mathsf{Obs}(R_s)} t$ and $\mathsf{head}(r\rho) \to^*_{\mathsf{Obs}(R_s)} t$. The argument for $\mathsf{tail}(\ell\rho)$ and $\mathsf{tail}(r\rho)$ is similar. □

Lemma 3. *The model* $(S_1, ([f]_1)_{f\in\Sigma_s})$ *satisfies the specification* $(\Sigma_d, \Sigma_s, R_d, R_s)$.

Proof. We have to prove that $[\ell\rho]_1(i) = [r\rho]_1(i)$ for every rule $\ell \to r$ in R_s, every ground substitution ρ and every $i \in \mathbf{N}$. By definition $[\ell\rho]_1(i)$ is the unique normal form with respect to $\mathsf{Obs}(R_s) \cup R_d$ of $\mathsf{head}(\mathsf{tail}^i(\ell\rho))$, and $[r\rho]_1(i)$ is the similar normal form of $\mathsf{head}(\mathsf{tail}^i(r\rho))$. Now the lemma follows from Lemma 2. □

For concluding the proof of Theorem 1 we have to prove that $(S_1, ([f]_1)_{f\in\Sigma_s})$ is the only model satisfying the specification $(\Sigma_d, \Sigma_s, R_d, R_s)$ and $S = \{[t] \mid t \in \mathcal{T}_s\}$. This follows from the following lemma.

Lemma 4. *Let* $(S, ([f])_{f\in\Sigma_s})$ *be any model satisfying* $(\Sigma_d, \Sigma_s, R_d, R_s)$, *and* $t \in \mathcal{T}_s$. *Then* $[t] = [t]_1$.

Proof. By definition in the model for $u \in D$ and $s \in S$ we have

$$([:](u, s))(0) = u, \quad ([:](u, s))(i) = s(i - 1) \text{ for } i > 0.$$

In the original stream specification the symbols $\mathsf{head}, \mathsf{tail}$ do not occur, for these fresh symbols we now define functions $[\mathsf{head}]$ and $[\mathsf{tail}]$ on streams s by

$$[\mathsf{head}](s) = s(0), \quad ([\mathsf{tail}](s))(i) = s(i + 1) \text{ for } i \geq 0.$$

If $S \neq D^\omega$ then it is not clear whether $[\mathsf{tail}](s) \in S$ for every $s \in S$. Therefore we extend S to D^ω and define $[f](\cdots)$ to be any arbitrary value if at least one argument is in $D^\omega \setminus S$; note that for the model satisfying the specification we only required $[\ell\rho] = [r\rho]$ for ground substitutions to $\mathcal{T}_s$ by which these junk values do not play a role.

Due to the definitions of $[:]$, $[\mathsf{head}]$ and $[\mathsf{tail}]$ this extended model satisfies the equations

$$\mathcal{E} = \begin{cases} \mathsf{head}(x : \sigma) = x \\ \mathsf{tail}(x : \sigma) = \sigma \\ \sigma = \mathsf{head}(\sigma) : \mathsf{tail}(\sigma) \end{cases}$$

that is, for ρ mapping x to any term of sort d and σ to any term of sort s we have $[\ell\rho] = [r\rho]$ for every $\ell \to r \in \mathcal{E}$. From the definition of $\mathsf{Obs}(R_s)$ it is easily checked that any innermost step $t \to_{\mathsf{Obs}(R_s)} t'$ on a ground term t is either an application of one of the first two rules of $\mathcal{E}$, or it is of the shape

$$t \to_{\mathcal{E}}^* \cdot \to_{R_s} \cdot \to_{\mathcal{E}}^* t'$$

where due to the innermost requirement the redex of the $\to_{R_s}$ step does not contain the symbols head or tail so is in $\mathcal{T}_s$. Since the model is assumed to satisfy the specification $(\Sigma_d, \Sigma_s, R_d, R_s)$, we conclude that $[t] = [t']$ for every innermost ground step $t \to_{\mathsf{Obs}(R_s)} t'$.

For the lemma we have to prove that $[t](i) = [t]_1(i)$ for every $i \in \mathbf{N}$. By definition $[t]_1(i)$ is the normal form with respect to $\mathsf{Obs}(R_s) \cup R_d$ of $\mathsf{head}(\mathsf{tail}^i(t))$.

Now consider an innermost $\mathsf{Obs}(R_s) \cup R_d$-reduction of $\mathsf{head}(\mathsf{tail}^i(t))$ to $[t]_1(i)$. By the above observation and the definitions of $[\mathsf{head}]$ and $[\mathsf{tail}]$ we conclude that

$$[t](i) = [\mathsf{head}(\mathsf{tail}^i(t))] = [[t]_1(i)] = [t]_1(i),$$

the last step since $[t]_1(i) \in D$. This concludes the proof, both of the lemma and Theorem 1. □

We conclude this section by an example of a well-defined stream specification that is not productive.

Example 4. Choose $\Sigma_s = \{c, f, g\}$, $\Sigma_d = \{0, 1\}$, $R_d = \emptyset$, and R_s consists of the following rules:

$$\begin{aligned} c &= 1 : c \\ f(x : \sigma) &= g(f(\sigma)) \\ g(x : \sigma) &= c. \end{aligned}$$

Then this is a valid stream specification for which $\mathsf{Obs}(R_s)$ is terminating, as can be shown by AProVE [4] or TTT2 [7]. Hence by Theorem 1 there is a unique model. So the ground term $f(c)$ has a unique interpretation: the stream only consisting of 1's. However, $f(c)$ is not productive.

So the TRS R_s is not suitable to compute the interpretation of $f(c)$. Instead one can use outermost reduction with respect to $\mathsf{P}(R_s)$, where $\mathsf{P}(R_s)$ is the TRS introduced in the definition of $\mathsf{Obs}(R_s)$.

5 Data Independent Stream Functions

The reason that in Theorem 1 we have to restrict to models satisfying $S = \{[t] \mid t \in \mathcal{T}_s\}$, as we saw in Example 3, is in the fact that computations may be guarded by data elements in left hand sides of rules. Next we show that we also get well-definedness for stream functions defined on all streams in case the left hand sides of the rule do not contain data elements.

Theorem 2. *Let $(\Sigma_d, \Sigma_s, R_d, R_s)$ be a stream specification for which the TRS $\mathsf{Obs}(R_s) \cup R_d$ is terminating and the only subterms of left hand sides of R_s of sort d are variables. Then the stream specification admits a unique model $(S, ([f])_{f \in \Sigma_s})$ satisfying $S = D^\omega$.*

Proof. (sketch) We have to prove that for any $f \in \Sigma_s$ of type $d^n \times s^m \to s$ the function $[f] : D^n \times (D^\omega)^m \to D^\omega$ is uniquely defined. For doing so we introduce m fresh constants $c_1, \ldots, c_m$ of sort s. Let $k \in \mathbf{N}$ and $u_1, \ldots, u_n \in D$. Due to termination and orthogonality of $\mathsf{Obs}(R_s) \cup R_d$, the term

$$\mathsf{head}(\mathsf{tail}^k(f(u_1, \ldots, u_n, c_1, \ldots, c_m)))$$

has a unique normal for with respect to $\mathsf{Obs}(R_s) \cup R_d$. Since it is of sort d, due to the shape of the rules it is a ground term of sort d over $\Sigma_d \cup \{\mathsf{head}, \mathsf{tail}, c_1, \ldots, c_m\}$, that is, a ground term T composed from Σ_d and terms of the shape $\mathsf{head}(\mathsf{tail}^i(c_j))$

for $i \in \mathbf{N}$ and $j \in \{1, \ldots, m\}$. For this observation it is essential that left hand sides do not contain non-variable terms of sort d: terms of the shape $f(\mathsf{head}(\cdots), \ldots)$ should be rewritten.

Let N be the greatest number i for which T has a subterm of the shape $\mathsf{head}(\mathsf{tail}^i(c_j))$. Let $s_1, \ldots, s_m \in D^\omega$. Define $t_j = s_j(0) : s_j(1) : \cdots : s_j(N) : \sigma$. Since $\mathsf{head}(\mathsf{tail}^k(f(u_1, \ldots, u_n, c_1, \ldots, c_m)))$ rewrites to T, the term $\mathsf{head}(\mathsf{tail}^k(f(u_1, \ldots, u_n, t_1, \ldots, t_m)))$ rewrites to T' obtained from T by replacing every subterm of the shape $\mathsf{head}(\mathsf{tail}^i(c_j))$ by $\mathsf{head}(\mathsf{tail}^i(t_j))$. Observe that $\mathsf{head}(\mathsf{tail}^i(t_j))$ rewrites to $s_j(i) \in D$. So $([f](u_1, \ldots, u_n, s_1, \ldots, s_m))(k)$ has to be the R_d-normal form of the ground term over Σ_d obtained from T by replacing every subterm of the shape $\mathsf{head}(\mathsf{tail}^i(c_j))$ by $s_j(i) \in D$. Since this fixes $([f](u_1, \ldots, u_n, s_1, \ldots, s_m))(k)$ for every k, this uniquely defines $[f]$. □

Example 5. It is easy to see that for the standard stream functions zip, even and odd defined by

$$\mathsf{even}(x : \sigma) = x : \mathsf{odd}(\sigma), \quad \mathsf{odd}(x : \sigma) = \mathsf{even}(\sigma), \quad \mathsf{zip}(x : \sigma, \tau) = x : \mathsf{zip}(\tau, \sigma),$$

there exists $f : D^\omega \to D^\omega$ for every data set D satisfying

$$f(x : \sigma) \;=\; x : \mathsf{zip}(f(\mathsf{even}(\sigma)), f(\mathsf{odd}(\sigma))),$$

namely the identity. By Theorem 2 we can conclude it is the only one, since for $R_d = \emptyset$ and R_s consisting of the above four rules, the resulting TRS $\mathsf{Obs}(R_s)$ is terminating as can be proved by AProVE [4] or TTT2 [7]. Both [3] and [11] fail to prove that the identity is the only stream function satisfying the equation for f. By essentially choosing $\mathsf{Obs}(R_s)$ as the input and adding information about special contexts, the tool Circ [8] is able to prove that f is the identity.

6 Fixpoints

Several streams are defined as fixpoints of stream functions, like the Fibonacci stream as given in the introduction. In our format it can be presented as the stream specification R_s consisting of the rules

$$\begin{array}{rl} \mathsf{Fib} = f(\mathsf{Fib}) & \qquad g(0, \sigma) = 0 : 1 : f(\sigma) \\ f(x : \sigma) = g(x, \sigma) & \qquad g(1, \sigma) = 0 : f(\sigma). \end{array}$$

The TRS $\mathsf{Obs}(R_s)$ is not terminating since it allows the reduction

$$\mathsf{head}(\mathsf{Fib}) \to \mathsf{head}(f(\mathsf{Fib})) \to \mathsf{head}(g(\mathsf{head}(\mathsf{Fib}), \mathsf{tail}(\mathsf{Fib}))).$$

However, now we will polish R_s to R'_s such that $\mathsf{Obs}(R'_s)$ is terminating, by which well-definedness of both R'_s and R_s can be concluded. This shows incompleteness of Theorem 1: the stream specification R_s admits a unique model but $\mathsf{Obs}(R_s)$ is not terminating.

Assume some model satisfies R_s; for simplicity we identify ground terms with their interpretations in the model. Then $\mathsf{Fib} = f(\mathsf{Fib}) = g(\cdots) = 0 : c$ for some stream c. Using this equality $\mathsf{Fib} = 0 : c$ we obtain

$$0 : c = \mathsf{Fib} = f(\mathsf{Fib}) = f(0 : c) = 0 : 1 : f(c),$$

so $c = 1 : f(c)$. So the model also satisfies R'_s which is obtained from R_s by replacing the first rule $\mathsf{Fib} = f(\mathsf{Fib})$ by the two rules $\mathsf{Fib} = 0 : c$ and $c = 1 : f(c)$. However, R'_s again satisfies our format and $\mathsf{Obs}(R'_s)$ is terminating as can be proved by AProVE [4] or TTT2 [7]. So by Theorem 1 R'_s admits a unique model, which is by construction the only model for R_s too.

This technique of modifying the stream specification is generally applicable. If our technique fails for proving well-definedness of a stream specification, we can analyze the specified streams by applying the rules and deriving new equalities from which a modified stream specification can be composed. If our technique succeeds in proving well-definedness of the modified specification, conclusions can be drawn about the original one.

In general, stream functions may have zero, one or several fixpoints. For instance, the boolean stream function f defined by

$$f(0 : \sigma) = 0 : 1 : f(\sigma), \qquad f(1 : \sigma) = 1 : 0 : f(\sigma),$$

has two fixpoints: the Thue Morse stream morse from Example 1 and its inverse. Proving that there are exactly two can be done as follows. Assume m is a fixpoint starting with 0, so $m = 0 : c$. Then $0 : c = m = f(m) = f(0 : c) = 0 : 1 : f(c)$, so $c = 1 : f(c)$. By adding the rules $m = 0 : c$ and $c = 1 : f(c)$ we have a stream specification R_s for which termination of $\mathsf{Obs}(R_s)$ can be proved. So there is exactly one fixpoint of f starting with 0, and by symmetry there is exactly one fixpoint of f starting with 1.

7 Conclusions

We presented a technique by which well-definedness of stream specifications like

$$\begin{aligned} f(0 : \sigma) &= 1 : f(\sigma) \\ f(1 : \sigma) &= 0 : f(f(\sigma) \\ c &= 1 : c \end{aligned}$$

can be proved fully automatically, where a tool like Circ [8] fails, and the productivity tool [3] fails to prove productivity of $f(c)$. The main idea is to prove well-definedness by proving termination of a transformed system $\mathsf{Obs}(R_s)$, in this way exploiting the power of present termination provers.

We observed that productivity of the stream specification can not be concluded from termination of $\mathsf{Obs}(R_s)$; we leave as a challenge to find syntactic criteria on the stream specification by which this can be concluded.

Acknowledgments. We want to thank Venanzio Capretta, Joerg Endrullis, Herman Geuvers, Jan Willem Klop, Dorel Lucanu, Matthias Raffelsieper, Grigore Rosu and Alexandra Silva for fruitful discussions on this exciting topic, and the anonymous referees for fruitful suggestions.

References

1. Allouche, J.-P., Shallit, J.: Automatic Sequences: Theory, Applications, Generalizations. Cambridge University Press, Cambridge (2003)
2. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoretical Computer Science 236, 133–178 (2000)
3. Endrullis, J., Grabmayer, C., Hendriks, D.: Data-oblivious stream productivity. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS, vol. 5330, pp. 79–96. Springer, Heidelberg (2008), webinterface tool: `http://fspc282.few.vu.nl/productivity/`
4. Giesl, J., et al.: Automated program verification environment (AProVE), `http://aprove.informatik.rwth-aachen.de/`
5. Giesl, J., Thiemann, R., Schneider-Kamp, P.: The dependency pair framework: Combining techniques for automated termination proofs. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS, vol. 3452, pp. 301–331. Springer, Heidelberg (2005)
6. Goguen, J., Lin, K., Rosu, G.: Circular coinductive rewriting. In: Proceedings, 15th International Conference on Automated Software Engineering (ASE 2000), Grenoble, France, September 11-15, 2000, Institute of Electrical and Electronics Engineers Computer Society (2000), webinterface tool CIRC: `http://fsl.cs.uiuc.edu/index.php/Special:CircOnline`
7. Korp, M., Sternagel, C., Zankl, H., Middeldorp, A.: Tyrolean termination tool 2. In: Treinen, R. (ed.) RTA 2009. LNCS, vol. 5595, pp. 295–304. Springer, Heidelberg (2009), Tool available at: `http://colo6-c703.uibk.ac.at/ttt2/`
8. Lucanu, D., Rosu, G.: CIRC: A circular coinductive prover. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 372–378. Springer, Heidelberg (2007)
9. Marche, C., Zantema, H.: The termination competition. In: Baader, F. (ed.) RTA 2007. LNCS, vol. 4533, pp. 303–313. Springer, Heidelberg (2007)
10. Roşu, G.: Equality of streams is a Π^0_2-complete problem. In: Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming (ICFP 2006). ACM Press, New York (2006)
11. Rutten, J.J.M.M.: A coinductive calculus of streams. Mathematical Structures in Computer Science 15, 93–147 (2005)
12. Simonsen, J.G.: The Π^0_2-completeness of most of the properties of rewriting systems you care about (and productivity). In: Treinen, R. (ed.) RTA 2009. LNCS, vol. 5595, pp. 335–349. Springer, Heidelberg (2009)

Modularity of Convergence in Infinitary Rewriting

Stefan Kahrs

University of Kent, Department of Computer Science, Canterbury CT2 7NF

Abstract. Properties of Term Rewriting Systems are called *modular* iff they are preserved under disjoint union, i.e. when combining two Term Rewriting Systems with disjoint signatures. *Convergence* is the property of Infinitary Term Rewriting Systems that all reduction sequences converge to a limit. *Strong Convergence* requires in addition that no redex position in a reduction sequence is used infinitely often.

In this paper it is shown that Strong Convergence is a modular property, lifting a restriction from a known result by Simonsen, and that Convergence is modular for non-collapsing Infinitary Term Rewriting Systems.

1 Introduction

Modular properties of Term Rewriting are properties that are preserved under (and reflected by) the disjoint union of signatures. They are of particular interest to reason about rewrite systems by *divide and conquer*. Examples of modular properties are: confluence, weak normalisation, simple termination; examples of non-modular properties are: strong normalisation, strong confluence.

One aspect of Term Rewriting is that it can be viewed as a computational model for Functional Programming. Lazy Functional Programming exhibits a phenomenon that ordinary Term Rewriting does not really capture: reductions that converge to an infinite result in infinitely many steps. Partly for that reason, the concept of Infinitary Term Rewriting was devised in [2]; technically, it arises from the finite version by equipping the signature with a metric d_m and using as its universe of terms $Ter^m(\Sigma)$, the metric completion of the metric space $(Ter(\Sigma), d_m)$. In this paper, we are only concerned with the metric d_∞ (which goes back to [1]) which sets the distance $d_\infty(t, t')$ to 2^{-k} where k is the length of the shortest position at which the two terms t and t' have different subterm-roots. This also coincides with a straightforward co-inductive definition of infinitary terms [7].

Infinitary Term Rewriting views transfinite reductions as reduction sequences that are also convergent sequences, w.r.t. its metric. An iTRS is called *convergent* iff all its reduction sequences converge.

There are few positive results about convergence in the literature, instead research has focussed on *strong convergence* which in addition requires that redex positions in a reduction sequence cannot stay short.

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 179–193, 2009.
© Springer-Verlag Berlin Heidelberg 2009

It is known that strong normalisation is not modular for finitary Term Rewriting Systems. The well-known counter example from [12] has a strongly normalising TRS with rule $F(0,1,x) \to F(x,x,x)$, and another one with rules $G(x,y) \to x$, $G(x,y) \to y$; their combination fails to be strong normalising as the term $F(0,1,G(0,1))$ reduces to itself in three steps. For the discrete metric strong convergence and strong normalisation coincide and therefore modularity of strong convergence can fail for other term metrics than d_∞. The example also shows that "finite convergence", the convergence of all reduction sequences starting from a finite term, fails to be modular in both weak and strong form [10].

This paper proves that strong convergence is a modular property, and that convergence is modular in the absence of collapsing rules. This generalises similar claims by Simonsen [10] about left-linear systems. Moreover, in the case of convergence his proof sketch has a gap as it assumes that the sequences in principal subterm positions of the combined system are eventually reduction sequences. This may be true for left-linear systems but it requires proof.

The modularity proofs work by contradiction, constructing a non-converging reduction sequence in one of the original systems from a non-converging one in the combined system. For strong convergence that proof strategy works directly. For convergence, this is much more complex (especially in the presence of non-left-linear rules): if a reduction sequence is converging but not strongly converging then the top layer "remains active" throughout the sequence and in particular it can rearrange its principal subterms. However, these rearrangements are strongly constrained by the fact that the iTRS associated with the top layer is itself converging. In particular, the subterms at a principal position in the limit form a so-called *focussed sequence*; these have peculiar properties: (i) any infinite subsequence of a focussed sequence contains an infinite reduction sequence; (ii) any non-converging focussed sequence contains a non-converging reduction sequence.

2 Basic Definitions and Results about Convergence

A *sequence* in a metric space (M,d) is a continuous function $f : \alpha \to M$, where α is an ordinal (w.r.t. the usual order topology). It is called *open* if α is a limit ordinal, otherwise it is *closed*.

An open sequence $f : \alpha \to M$ *diverges* iff it cannot be extended to a sequence $f' : \alpha + 1 \to M$ where $\forall \gamma < \alpha.\ f'(\gamma) = f(\gamma)$.

A *subsequence* of f is a strictly monotonic and continuous function $g : \beta \to \alpha$. g is called *cofinal* [3] iff $\forall \gamma.\ \gamma < \alpha \Rightarrow \exists \zeta.\zeta < \beta.\ g(\zeta) \geq \gamma$. We say that a subsequence g converges iff the sequence $f \circ g$ does. Clearly, an open sequence converges iff all its subsequences do. Note: (i) cofinal subsequences compose: if g is a cofinal subsequence of f and h a cofinal subsequence of g then $g \circ h$ is a cofinal subsequence of f; (ii) subsequences are *strictly normal* functions on ordinals [8], i.e. their application distributes over suprema.

A sequence f converges to c iff it stays eventually within any neighbourhood of c. Thus, if f does not converge to c then there is a neighbourhood B of c and a cofinal subsequence g of f such that B and the range of $f \circ g$ are disjoint.

A *metric abstract reduction system* (short: MARS) is a structure $(M, d, \to)$ such that (M, d) is a metric space and $(M, \to)$ an abstract reduction system, i.e. $\to$ is a binary relation on M. Aside: despite sharing the acronym, this definition differs from the one originally used by Kennaway [5], as that requires additional structure.

A reduction sequence in a MARS $(M, d, \to)$ is a sequence $f : \alpha \to M$ such that $\forall \beta.\beta + 1 < \alpha \Rightarrow f(\beta) \to f(\beta + 1)$. Notice that being a sequence entails the continuity of f.

Proposition 1. *If the open sequence $f : \alpha \to M$ diverges and (M, d) is a complete metric space then there is an $\epsilon > 0$ such that $\forall \beta < \alpha.\ \exists \gamma, \gamma'.\ \beta \leq \gamma < \gamma' < \alpha \wedge d(f(\gamma), f(\gamma')) \geq \epsilon$.*

Proof. Suppose there was no such ϵ. We can construct a Cauchy-sequence a_n by choosing $a_n = f(\beta_n)$ where β_n is the smallest ordinal such that the f-image of $[\beta_n, \alpha)$ is a set with diameter no bigger than $1/n$. Completeness of (M, d) means that a_n has a limit a, and we can extend f to f' with $f'(\alpha) = a$ as the condition also ensures that f' is continuous at α. Thus f did not diverge. □

A MARS is called *convergent* iff all its open reduction sequences converge. Otherwise we call it *divergent*.

Weak reduction $\twoheadrightarrow_w$ of a MARS is defined as follows: $t \twoheadrightarrow_w u$ iff there exists a closed reduction sequence $f : \alpha + 1 \to M$ with $f(0) = t$ and $f(\alpha) = u$. Clearly, the relation $\twoheadrightarrow_w$ is transitive. We also write $\twoheadrightarrow_{ww}$ for the weak reduction of the MARS $(M, d, \twoheadrightarrow_w)$. Moreover, we call reduction sequences of the MARS $(M, d, \twoheadrightarrow_w)$ *weak reduction sequences* of the original MARS $(M, d, \to)$.

Proposition 2. *Let $(M, d, \to)$ be a convergent MARS.*
If $t \twoheadrightarrow_{ww} u$ then $t \twoheadrightarrow_w u$.

Proof. We can prove the result by induction on the indexing ordinal α of the witnessing sequence $f : \alpha + 1 \to M$ for $t \twoheadrightarrow_{ww} u$. The base case $\alpha = 0$ is trivial and if α is a successor ordinal the result follows by the induction hypothesis and transitivity of $\twoheadrightarrow_w$.

Otherwise, α is a limit ordinal. For any $\beta < \alpha$ we have $t \twoheadrightarrow_{ww} f(\beta)$ and thus by the induction hypothesis $t \twoheadrightarrow_w f(\beta)$. This reduction has an associated indexing function $g_\beta : \gamma_\beta \to M$. The increasing sequence of ordinals γ_β (with β approaching α) must converge to some ordinal γ [9, page 290]. W.l.o.g. we can assume that this is a limit ordinal as otherwise almost all reductions in the image of f would be empty. Moreover the functions g_β agree on their common domain. In their limit they thus extend to a function $g : \gamma \to M$ which is an open reduction sequence. As the original MARS is convergent g can be extended to a closed sequence $g' : \gamma + 1 \to M$; since both $f(\alpha)$ and $g'(\gamma)$ are limits of the sequence $f(\beta)$ with β approaching α we must have $g'(\gamma) = f(\alpha)$ and the result follows. □

It is easiest to explain the meaning of Proposition 2 by showing how the property fails if we drop the condition that $(M, d, \to)$ is convergent. Consider the following example, given as an infinitary string rewriting system:

$$\{BE \to CSE,\ AC \to AB,\ BS \to SB,\ SC \to CS\}$$

In this system we have for any n:

$$ABS^nE \to^* AS^nBE \to AS^nCSE \to^* ACS^{n+1}E \to ABS^{n+1}E$$

Thus $ABS^nE \to^* ABS^{n+1}E$ and also $ABS^nE \twoheadrightarrow_w ABS^{n+1}E$. Starting from $n = 0$ we get $ABE \twoheadrightarrow_{ww} ABS^\infty$ as the result is the limit of all ABS^nE. However, we do not have $ABE \twoheadrightarrow_w ABS^\infty$ as intermediate reduction results in the sequences for $ABS^nE \twoheadrightarrow_w ABS^{n+1}E$ will change the B to an S. The example does not contradict the proposition as the system is not converging: we have $ABS^nE \twoheadrightarrow_w AS^nBE \twoheadrightarrow_w ABS^{n+1}E \twoheadrightarrow_w \ldots$.

Aside: the converse does not hold, i.e. it is possible that $\twoheadrightarrow_w$ and $\twoheadrightarrow_{ww}$ coincide without $(M, d, \to)$ being convergent — e.g. we can choose $\to = M \times M$.

Proposition 3. *Let $(M, d, \to)$ be a convergent MARS.*
Then $(M, d, \twoheadrightarrow_w)$ is also convergent.

Proof. Using the same construction as in the proof of Proposition 2 we can expand an open $\twoheadrightarrow_w$-reduction into an open $\to$-reduction. The convergence assumption gives us a limit to the latter which must also be a limit to the former. □

Corollary 1. *A MARS is convergent if and only if all its weak reduction sequences converge.*

Proof. If the MARS is not convergent then the result follows because any reduction sequence is also a weak reduction sequence. Otherwise Proposition 3 applies. □

Given a MARS $(M, d, \to)$ and a sequence $f : \alpha \to M$, the predicate $\mathrm{Foc}_{f,\alpha}$ on α is defined as follows:

$$\mathrm{Foc}_{f,\alpha}(\beta) \iff \forall\gamma.\, \alpha > \gamma \geq \beta \Rightarrow \exists\zeta.\, \alpha > \zeta \wedge \forall\kappa.\, \alpha > \kappa \geq \zeta \Rightarrow f(\gamma) \twoheadrightarrow_w f(\kappa)$$

We have: (i) if $\mathrm{Foc}_{f,\alpha}(\beta)$ then $\mathrm{Foc}_{f,\alpha}(\gamma)$ for all $\gamma \geq \beta$; (ii) if $\mathrm{Foc}_{f,\alpha}(\beta)$ then $\exists\gamma > \beta.\, f(\beta) \twoheadrightarrow_w f(\gamma)$. The sequence f is called *focussed* iff $\exists\beta.\, \mathrm{Foc}_{f,\alpha}(\beta)$.

In words: a sequence is focussed iff every of its elements sufficiently close to α weakly reduces to all its elements sufficiently close to α. In particular, all weak reduction sequences are focussed, but not vice versa.

Of interest are specifically *open* focussed sequences. If a focussed sequence is *closed*, i.e. if $\alpha = \alpha' + 1$ for some α' then because of the possible choice $\kappa = \alpha'$, the condition $\mathrm{Foc}_{f,\alpha}(\beta)$ can be simplified to: $\forall\gamma.\, \alpha > \gamma \geq \beta \Rightarrow f(\gamma) \twoheadrightarrow_w f(\alpha')$, i.e. every sufficiently late occurring element of the sequence weakly reduces to the last element.

Proposition 4. *Every cofinal subsequence g of a focussed sequence f has itself a cofinal subsequence h such that $f \circ g \circ h$ is a cofinal weak reduction sequence.*

Proof. First notice that this is trivial if f is closed, because g would be forced to include the last element of f (as g is cofinal), and it would suffice if h is singleton, picking that last element. Otherwise, the input domain of f is a limit ordinal α.

Let $f : \alpha \to M$ be focussed and let $g : \mu \to \alpha$ be the cofinal subsequence. We construct a strictly monotonic $h' : \alpha \to \mu + 1$ as follows.

$$\begin{aligned} h'(0) &= \min\{\nu \mid \mathrm{Foc}_{f,\alpha}(g(\nu))\} \\ h'(\gamma+1) &= \min\{\zeta \mid \zeta > h'(\gamma) \wedge f(g(h'(\gamma))) \twoheadrightarrow_w f(g(\zeta))\} \\ h'(\lambda) &= \lim_{\gamma<\lambda} h'(\gamma) \end{aligned}$$

If for some limit ordinal $\lambda < \alpha$ we have $h'(\lambda) = \mu$ then $h = h'|_\lambda$. Otherwise μ does not appear in the range of h' and we can set $h = h'$.

The focussed property ensures that the minima on the right-hand sides are minima of non-empty sets, which both implies that h is strictly increasing and that $f \circ g \circ h$ is a reduction sequence.

To show that the map h is cofinal two cases need to be considered. First, if its domain is $\lambda \neq \alpha$ then that was because $\mu = h'(\lambda) = \lim_{\gamma<\lambda} h'(\gamma) = \lim_{\gamma<\lambda} h(\gamma)$, showing that h is cofinal. Otherwise, $h = h'$ and its domain is α. Assume $\lim_{\gamma<\alpha} h(\gamma) = \mu' < \mu$. Then, using basic facts about strictly normal functions on ordinals [8,3], we can derive a contradiction:

$$\alpha = \lim_{\gamma<\alpha} \gamma \leq \lim_{\gamma<\alpha} g(h(\gamma)) = g(\lim_{\gamma<\alpha} h(\gamma)) = g(\mu') < \alpha$$

Thus, the assumption was wrong and h is cofinal. The subsequence $g \circ h$ is also cofinal, because cofinal maps are preserved by composition. □

One could say that within a focussed sequence the $\twoheadrightarrow_w$ relation is like an *almost full* relation [11, page 811]. What complicates the situation though is that we are dealing with transfinite sequences. In particular, to relate the convergence of f to that of one of its subsequences we need that the subsequence is cofinal; without that property the subsequence could converge to a limit that has no bearings on the overall convergence.

For example, given rules $A \to S(A)$, $B \to S(B)$, $F(x,x) \to S(F(x,x))$ we have an open reduction $f : \omega+\omega \to \mathit{Ter}^\infty(\Sigma)$ commencing in $F(A,B)$ and converging to S^∞. f contains an infinite subsequence that converges to $F(S^\infty, S^\infty)$, but this subsequence is not cofinal. Any cofinal subsequence of f would also converge to S^∞ .

Proposition 5. *A MARS converges iff all its focussed sequences converge.*

Proof. Since all weak reduction sequences are focussed the $\Leftarrow$ implication follows from Corollary 1. Now suppose $f : \alpha \to M$ is a divergent focussed sequence; from it, we need to construct a divergent reduction sequence, and by Corollary 1 it suffices if it is a weak reduction sequence.

We can apply Proposition 4 to f, using the identity function $id : \alpha \to \alpha$ as the cofinal subsequence. This gives us a cofinal subsequence $h_c : \beta \to \alpha$ such that $g_c = f \circ h_c$ is a weak reduction sequence.

If g_c is divergent we are done. Otherwise it converges to a limit c. Since f diverges it does not converge to c. Thus there is a proper subsequence s of f

such that the values in the range of $f \circ s$ are outside some neighbourhood B of c. Again, we can apply Proposition 4, this time to f and s. This gives us a cofinal subsequence $h_e : \beta' \to \alpha$ such that $g_e = f \circ h_e$ is a weak reduction sequence.

If g_e is divergent we are done. Otherwise it converges to a limit $e \neq c$. Then we can construct a third sequence $h : \alpha \to \alpha + 1$ as follows:

$$\begin{aligned} h(0) &= h_c(0) \\ h(\gamma+1) &= \min\{\zeta > h(\gamma) \mid h_\gamma(\zeta') = \zeta \wedge f(h(\gamma)) \twoheadrightarrow_w f(\zeta)\} \\ h(\lambda) &= \lim_{\gamma<\lambda} h(\gamma) \end{aligned}$$

In the second equation the function h_γ is either h_c or h_e, depending on whether γ is even or odd. As in the proof of Proposition 4 we obtain the required cofinal reduction subsequence by constraining the domain of h to λ, if necessary. That sequence cannot converge because the distance between subsequent elements converges to $d(c, e)$. □

Note: the proof could be generalised to reduction systems over topological spaces that are Hausdorff, as the final argument only rests upon that c and e have disjoint neighbourhoods. Whether the result could be generalised further to topological spaces that are not Hausdorff is not clear.

3 Infinitary Term Rewriting

For the basic definitions of infinitary term rewriting we refer to [6]. Summarised briefly: the set $\mathit{Ter}^\infty(\Sigma)$ of infinite terms arises as the metric completion of the metric space $(\mathit{Ter}(\Sigma), d_\infty)$, where the metric d_∞ is inductively defined on finite terms as $d_\infty(t, u) = 1$ if t and u have different roots, and otherwise $d_\infty(F(t_1, \ldots, t_n), F(u_1, \ldots, u_n)) = \frac{1}{2} \cdot \max_{1 \leq i \leq n}(d_\infty(t_i, u_i))$. Rewrite rules have the form $l \to r$ where l is a non-variable term and all variables in r occur in l; usually, infinite left-hand sides are not permitted in rules, but there is no need to exclude them for the purposes of this paper.

The notions of contexts and substitutions (and their applications) can be lifted in a canonical way from their finite counterparts [4]. An *inifinitary Term Rewriting System* (iTRS) is given by a signature Σ and a set of rewrite rules R. The single-step reduction relation $\to_R$ arises as usual by closing R under substitution application and context application.

For selecting the subterm at position p of a term t we use the notation t/p. We totalise this partial function by setting $t/p = x$ if $p \notin \mathsf{Pos}(t)$ which allows to extend this notation pointwise to sequences: $(f/p)(\alpha) = f(\alpha)/p$.

Proposition 6. *Let $(\mathit{Ter}^\infty(\Sigma), d_\infty, \to_R)$ be the MARS of an iTRS R. If $f : \alpha \to \mathit{Ter}^\infty(\Sigma)$ is a divergent sequence then there is a position p such that $\forall \beta < \alpha.\, \exists \gamma, \gamma'.\, \beta \leq \gamma < \gamma' < \alpha \wedge d_\infty(f(\gamma)/p, f(\gamma')/p) = 1$.*

Proof. As $\mathit{Ter}^\infty(\Sigma)$ is complete Proposition 1 applies. Because all distances in $\mathit{Ter}^\infty(\Sigma)$ are negative powers of 2 there are only finitely many distances greater

or equal than any ϵ and thus there is a smallest n such that $d_\infty(f(\gamma), f(\gamma')) = 2^{-n}$ occurs for γ and γ' arbitrarily close to α. This means that all subterm positions q with $|q| < n$ are eventually fixed. However, once they are fixed there are only finitely many positions of length n. At least one of those must be affected in any neighbourhood of α and this is the mentioned p. □

This result is specific to metric d_∞ (modulo homeomorphism). The reason is this: any metric d_m is determined by its behaviour on finite terms. Any Cauchy-sequence in $(Ter(\Sigma), d_m)$ is also Cauchy in $(Ter(\Sigma), d_\infty)$; if the converse holds then $Ter^m(\Sigma)$ and $Ter^\infty(\Sigma)$ are homeomorphic; any counterexample would also be a counterexample to the proposition. This claim only assumes that other metrics d_m are defined as *term metrics* [4], i.e. by induction on the term structure.

An iTRS is *strongly converging* iff for all open reduction sequences of length α no redex position p is reduced arbitrarily close to α. Equivalent to this [13] is the notion of top-termination, i.e. that no reduction sequence contracts a root redex infinitely often.

It is possible to define the property without referring to redex positions though: the *indirected version* of an iTRSs R is defined as follows. Its signature is that of R extended by a fresh unary function symbol I; for each rule $l \to r$ in R it has a rule $l \to I(r)$; in addition, it has the rule $I(x) \to x$. We modify the metric d_∞ to set $d_\infty(I(t), I(u)) = d_\infty(t, u)$.

Proposition 7. *An iTRS is* strongly converging *iff its indirected version is converging.*

Proof. Any reduction sequence f of the original iTRS can be mapped into a reduction sequence f' of the indirected version by splitting the step into two parts, where the second part removes the just introduced I-symbol. If f is not top-terminating then f' fails to converge, as it would have infinitely many changes of root-symbol.

Dually, a reduction sequence g of the indirected version can be mapped into a (shorter) sequence g^+ of the original system, by removing all I-symbols. However, g can only fail to converge by an infinitely often occurring function symbol change at some minimal position p. That means though that g must apply rules other than $I(x) \to x$ at position p infinitely often, and thus g^+ would apply those rules at p or higher infinitely often which means that g^+ is not strongly converging. □

4 Counterexamples and Near Counterexamples

In this section we will be looking at examples that show why the modularity results for convergence come with side conditions.

Particulary problematic w.r.t. convergence are collapsing rules, i.e. rules that have a variable as their right-hand side. An iTRS immediately fails to be strongly convergent if a collapsing rule is present. The reason is this: any such rule takes the form $C[x, \ldots, x] \to x$, the equation $t = C[t, \ldots, t]$ has a (unique) solution in

$Ter^\infty(\Sigma)$, and $t \xrightarrow{\lambda} t$ gives an infinite reduction sequence with contractions at the root.

The weaker notion of convergence can coexist with collapsing rules, but barely: if $C[x] \to^* x$ holds for any other contexts $C[\ ]$ than those of the form $C[y] = F^n(y)$ (for varying n but fixed F) then such an iTRS is not convergent either. In particular, the presence of two rules $F(x) \to x$ and $G(x) \to x$ means that the terms $t = F(G(t)), u = G(F(u))$ reduce to each other and thus give rise to a non-converging reduction sequence. The example also shows that convergence (on its own) is not a modular property.

Convergence can also be broken if only one of the two systems has a collapsing rule. Example: system R has rule $G(H(x)) \to G(x)$, system S has the rule $F(x) \to x$. In the combined system there is the term $t = F(H(t))$ and the non-convergent reduction $G(t) \to G(H(t)) \to G(t)$.

Most of the time we can observe convergence (or its absence) at ω-indexed reduction sequences. In the presence of non-left-linear rules this can be rather different though. Example [13]:

$$\{E \to S(E),\ F \to S(F),\ G(x,x) \to G(E,F)\}$$

This fails to converge, but this failure first occurs at ω^2-indexed reduction sequences. We have $G(E,F) \to^* G(S^n(E), S^m(F))$ for any m, n and thus in the limit $G(E,F) \twoheadrightarrow_w G(S^\infty, S^\infty) \to G(E,F)$. This does not provide any convergence problems at ordinals of the form $\omega \cdot k$ though; however, when we attempt to extend a reduction sequence to ω^2 we find both $G(E,F)$ and $G(S^\infty, S^\infty)$ in the image of the reduction sequence at any of its neighbourhoods.

In the previous example we had $E \to^* S^n(E)$ for any n, and $E \twoheadrightarrow_w S^\infty$ in the limit. However, one can have an iTRS with the former but not the latter property, e.g. if we replace the first rule by these four:

$$\{E \to Z,\ E \to H(E),\ H(Z) \to S(Z),\ H(S(x)) \to S(S(x))\}$$

Patterns like this complicate the reasoning about (weak) convergence, because in a combined system an outer layer may place in a position p different reducts of E, and thus the term in position p could converge at a limit to S^∞, showing that a principal subterm at a limit may not be a reduct of an earlier principal subterm. For example, this happens if we combine this system with the following:

$$\{J(K(x,y)) \to J(y)\}$$

Let $t = K(E,t)$ and $u = K(S^\infty, u)$: we have $J(t) \twoheadrightarrow_w J(u)$; E is the only principal subterm of $J(t)$ and it does not reduce to any of the principal subterms of $J(u)$. However, this second iTRS already fails to converge: given the term $s = K(J(K(x,y)), s)$ we can construct the non-converging sequence $J(s) \to J(K(J(y)), s) \to J(s)$.

5 Definitions and Observations Useful for Both Proofs

The first definition is taken from [4]: A family of equivalence relations $\overset{p}{\sim}$ (indexed by positions p) is inductively defined as follows:

$$t \overset{\lambda}{\sim} u$$
$$F(t_1, \ldots, t_n) \overset{i \cdot q}{\sim} F(u_1, \ldots, u_n) \Longleftarrow t_i \overset{q}{\sim} u_i$$

When studying the disjoint union of term rewriting systems R and S it is useful to split a term into *layers* of the participating systems, and distinguish its *principal subterms*. The principal subterm positions of a term t, $\mathsf{PPos}(t) \subset \mathsf{Pos}(t)$ have the following properties: p is principal if (i) the root of t/p is a function symbol belonging to a different signature than the root of t; (ii) no proper prefix of p is principal. The top positions of a term t, $\mathsf{TPos}(t) \subset \mathsf{Pos}(t)$ are defined as follows: $\mathsf{TPos}(t) = \{p \in \mathsf{Pos}(t) \mid \forall q.\, q \prec p \Rightarrow q \notin \mathsf{PPos}(t)\}$. Thus top positions are either parallel to principal subterms or their prefixes.

The top layer of a term is comprised of all the positions reachable from the root without a change of signature, and is thus "cut off" at its principal subterm positions. This leaves the layer with gaps at the cut-off points; therefore, the top layer is technically defined as the function that recreates a term if suitable fill material for the gaps is provided: The *top-layer* of a term t is a function $\lceil t \rceil$ with type $(\mathsf{PPos}(t) \to \mathit{Ter}^\infty(\Sigma)) \to \mathit{Ter}^\infty(\Sigma)$ where we write $\lceil t \rceil_\xi$ for the application of $\lceil t \rceil$ to a function $\xi : \mathsf{PPos}(t) \to \mathit{Ter}^\infty(\Sigma)$. The top layer has the following properties: above principal subterm positions it agrees with t:

$$\forall \xi.\, \forall p \in \mathsf{PPos}(t).\, t \overset{p}{\sim} \lceil t \rceil_\xi$$

In positions parallel to principal subterms, t and $\lceil t \rceil$ coincide:

$$\forall \xi.\, \forall q \in \mathsf{Pos}(t).\, (\forall p \in \mathsf{PPos}(t).\, q \| p) \Rightarrow t/q = \lceil t \rceil_\xi / q$$

And in principal subterm positions themselves the gaps are filled with the function ξ:

$$\forall p \in \mathsf{PPos}(t).\, \lceil t \rceil_\xi / p = \xi(p).$$

The standard notation [11] for layers is $C[\![t_1, \ldots, t_n]\!]$. The main reason for departing from that is that the top-layer of an infinitary term may have infinitely many principal subterm positions, and the $\lceil t \rceil$ notation avoids the need for accessing them through an indexing set by using $\mathsf{PPos}(t)$ itself instead.

The (possibly infinite) rank of an infinitary term t is defined as follows:

$$\mathit{rank}(t) = \sup\{1 + \mathit{rank}(t/p) \mid p \in \mathsf{PPos}(t)\}$$

We can define a distance between top-layers: $d(\lceil t \rceil, \lceil u \rceil) = d_\infty(\lceil t \rceil_{kx}, \lceil u \rceil_{kx})$ where kx is the function that constantly returns x, where x is a fresh variable.

From now the reasoning will be about the disjoint union of two non-collapsing iTRSs R and S with rewrite relation $\to_{RS}$ and combined signature Σ, and for

the top layer of any reduction (step or sequence) it is assumed that it is situated in system R, with signature Σ_R.

In the following we will be using a construction in which all principal subterms of a term t at finite rank n are replaced by a fixed term u, $t[n \searrow u]$. Formally:

$$t[0 \searrow u] = u$$
$$t[n+1 \searrow u] = \lceil t \rceil_\xi \quad \text{where } \xi(p) = t/p[n \searrow u]$$

This notation is also extended to sequences of terms: if $f : \alpha \to \mathit{Ter}^\infty(\Sigma)$ is continuous then $f[n \searrow u] : \alpha \to \mathit{Ter}^\infty(\Sigma)$ is defined pointwise as

$$f[n \searrow u](\beta) = f(\beta)[n \searrow u]$$

Notice that the function $t \mapsto t[n \searrow u]$ is non-expansive [4] and therefore preverses the continuity of the sequence.

Lemma 1. *Let $f : \alpha \to \mathit{Ter}^\infty(\Sigma)$ be a reduction sequence. Let $u \in \mathit{Ter}^\infty(\Sigma)$ and $n \in \mathcal{N}$ be arbitrary. Then the sequence $f[n \searrow u]$ is a reduction sequence of the reflexive closure of $\to_{RS}$.*

Proof. The absence of collapsing rules means that principal subterm replacement at level n commutes with reduction steps at higher level, even for contraction of non-left-linear redexes. Limits are also preserved by continuity of the construction. Reduction steps at lower level become reflexive steps. □

Any reduction sequence of the MARS $(M, d, \to^=)$ can be turned into a (possibly shorter) reduction sequence of $(M, d, \to)$. Thus, the presence of reflexive steps in $f[n \searrow u]$ is merely a technicality. The full generality of lemma 1 will be used later for lemma 6; of special interest is the case $n = 1$:

Corollary 2. *Let $f : \alpha \to \mathit{Ter}^\infty(\Sigma)$ be a reduction sequence. Let $u \in \mathit{Ter}^\infty(\Sigma_R)$. Then $f[1 \searrow u]$ is a reduction sequence of the reflexive closure of $\to_R$.*

Proof. As all terms of the sequence are by construction Σ_R-terms each $\to_{RS}$-step is a $\to_R$-step. □

over the signature of We define the principal positions of a reduction sequence $f : \alpha \to \mathit{Ter}^\infty(\Sigma)$ as follows:

$$\mathsf{PPos}(f) = \bigcup_{\beta<\alpha} \bigcap_{\beta<\gamma<\alpha} \mathsf{PPos}(f(\gamma))$$

Lemma 2. *Let $f : \alpha \to \mathit{Ter}^\infty(\Sigma)$ be an open reduction sequence. If R is convergent then the top layers of f converge to some top layer $\lceil u \rceil$ where $\mathsf{PPos}(u) = \mathsf{PPos}(f)$.*

Proof. Since the iTRS of the top layer is convergent then so is $f[1 \searrow x]$, for some fresh variable x, and let its limit be t. Let c be any non-variable term in the other iTRS then by substitutivity of weak convergence we also have $f[1 \searrow x][c/x] = f[1 \searrow c] \twoheadrightarrow_w t[x/c]$, and we can set $u = t[x/c]$ since $\lceil u \rceil$ does not depend on the choice of c. Clearly, any principal position in $t[x/c]$ must be principal for $f(\beta)[1 \searrow c]$ for all γ in the open interval (β, α) for some β. □

6 Modularity of Convergence

In the example section we have seen how a top-layer can through rearrangement of its principal subterms interfere with the convergence at subterm positions. However, the example that rearranged principal subterms "in a substantial way" was divergent.

We will show that this is not an accident. The idea behind this is the following: if f is an open reduction sequence then so is $g = f[1 \searrow l]$ and if l is a term belonging to the (converging) top-layer then g must converge. Moreover, if $l \to r$ (in the interesting case $l \neq r$) then g must still converge if we interleave it with the occasional reduction of $l \to r$, provided we preserve non-left-linear redexes along the way. It is impossible to have both l and r occur infinitely often in a fixed position p, because that would create a divergence with diameter of at least $d_\infty(l, r) \cdot 2^{-|p|}$. That means that two situations are ruled out: (i) that the top-layer reduction drags eventually infinitely many different subterms into position p — as in $f[1 \searrow l]$ each l could be reduced once it appears in position p; (ii) at the same time infinitely many terms appearing in position p are "descendants" of a principal subterm t and infinitely many others are not. In that case we would create a non-converging sequence by reducing the l in the corresponding place to t in $f[1 \searrow l]$, as all the terms corresponding to descendants would change to r while the others stay l. With both these scenarios ruled out one can show that the sequence of subterms in position p must converge.

These ideas will be formalised in a slightly different way that avoids the tracing of redexes and the rather cumbersome notion of "descendant" — which is problematic anyway for non-strong reductions [10].

A *predicate sequence* is an ordinal-indexed sequence of predicates s on terms such that $\beta \leq \gamma \wedge t \twoheadrightarrow_w u \wedge s(\gamma)(u) \Rightarrow s(\beta)(t)$. The idea is to use a predicate sequence alongside a reduction sequence of the same length.

Let $f : \alpha \to \mathit{Ter}^\infty(\Sigma)$ be a reduction sequence and p be an ω-sequence of positive integers. The function $f_p : \alpha \to \mathit{Ter}^\infty(\Sigma) \to \mathcal{B}$ maps ordinals to predicates on terms, and is defined as follows:

$$f_p(\beta)(t) \iff \exists \gamma.\, \beta \leq \gamma < \alpha \wedge p \in \mathsf{PPos}(f(\gamma)) \wedge t \twoheadrightarrow_w f(\gamma)/p$$

The predicate f_p is true for terms that weakly reduce to terms appearing in principal position p later in the sequence.

Lemma 3. *f_p is a predicate sequence.*

Proof. Assume $\beta \leq \gamma$ and $t \twoheadrightarrow_w u$ and $f_p(\gamma)(u)$. $f_p(\gamma)(u)$ means that there is a $\zeta \geq \gamma$ such that p is a principal position in $f(\zeta)$ and $u \twoheadrightarrow_w f(\zeta)/p$. In order to show that $f_p(\beta)(t)$ we can choose the same ζ as the witness; $\beta \leq \zeta$ holds because $\beta \leq \gamma \leq \zeta$; $t \twoheadrightarrow_w f(\zeta)/p$ holds because $t \twoheadrightarrow_w u \twoheadrightarrow_w f(\zeta)/p$, and $\twoheadrightarrow_w$ is transitive. □

Another predicate sequence we shall be using is not dependent on the ordinal, and defined as: $k_t(\beta)(u) \iff \neg(t \twoheadrightarrow_w u)$. Thus k_t is constantly true for all terms that are not reducts of t.

Lemma 4. *k_t is a predicate sequence.*

Proof. Let $r \twoheadrightarrow_w s$ and $k_t(\gamma)(s)$. Thus $\neg(t \twoheadrightarrow_w s)$. To show $k_t(\beta)(r)$ we need to show $\neg(t \twoheadrightarrow_w r)$. But $t \twoheadrightarrow_w r$ gives $t \twoheadrightarrow_w r \twoheadrightarrow_w s$ and so by transitivity of $\twoheadrightarrow_w$ the contradiction $t \twoheadrightarrow_w s$. □

Given a rule $l \to r$ in R and a predicate sequence s we define a dependent function $\xi_s : \Pi t : Ter^\infty(\Sigma).\alpha \to \mathsf{PPos}(t) \to Ter^\infty(\Sigma)$ as follows:

$$\xi_s(t)(\beta)(p) = \begin{cases} l & \text{if } s(\beta)(t/p) \\ r & \text{otherwise} \end{cases}$$

Proposition 8. *Distributive properties of ξ_s w.r.t. subterm selection and substitution application in top positions:*

$$\begin{aligned} \forall p \in \mathsf{TPos}(t).\ \lceil t \rceil_{\xi_s(t)(\beta)}/p &= \lceil t/p \rceil_{\xi_s(t/p)(\beta)} \\ \forall u \in Ter^\infty(\Sigma_R).\ \lceil \sigma(u) \rceil_{\xi_s(\sigma(u))(\beta)} &= \sigma'(u) \\ \text{where } \forall x.\ \sigma'(x) &= \lceil \sigma(x) \rceil_{\xi_s(\sigma(x))(\beta)} \end{aligned}$$

Proof. The first property is trivially proven by induction on the length of p. The second is a corollary, because if $u \in Ter^\infty(\Sigma_R)$ (i.e. only using function symbols from the signature of the top layer) then all variable positions in u become top positions in $\sigma(u)$. □

The function ξ_s is used to create reducts of $t[1 \searrow l]$ by reducing some of the l to r, guided by the predicate sequence. In particular, it is used to replace principal subterms in a reduction sequence.

Proposition 9. *The fundamental properties of the function ξ_s are:*

$$\begin{aligned} t \to_{RS} u \Rightarrow \lceil t \rceil_{\xi_s(t)(\beta)} &\to_R^{=} \lceil u \rceil_{\xi_s(u)(\beta)} \\ \beta \preceq \gamma \Rightarrow \lceil t \rceil_{\xi_s(t)(\beta)} &\twoheadrightarrow_w \lceil t \rceil_{\xi_s(t)(\gamma)} \end{aligned}$$

Proof. First property: let q be the redex position. At positions p parallel to q, $t \overset{p}{\sim} u$ thus also $\lceil t \rceil_{\xi_s(t)(\beta)} \overset{p}{\sim} \lceil u \rceil_{\xi_s(u)(\beta)}$.

If $p \preceq q$ for some $p \in \mathsf{PPos}(t)$ then certainly $p \in \mathsf{PPos}(u)$ and $\lceil t \rceil_{\xi_s(t)(\beta)}$ and $\lceil u \rceil_{\xi_s(u)(\beta)}$ also coincide in all positions that are prefixes of p. At position p itself these two terms have either l or r. If $\xi_s(u)(\beta)(p) = r$ then there is nothing to show, as both $l \to_R^{=} r$ and $r \to_R^{=} r$. If $\xi_s(u)(\beta)(p) = l$ then we know by the definition of ξ_s that $s(\beta)(u/p)$ holds. Since we know $t/p \to_{RS} u/p$ we can use the predicate sequence property of s to establish $s(\beta)(t/p)$. This means by the definition of ξ_s that $\xi_s(t)(\beta)(p) = l$, and therefore $\lceil t \rceil_{\xi_s(t)(\beta)} = \lceil u \rceil_{\xi_s(u)(\beta)}$ as the terms also agree in position p.

Otherwise the redex q is a top position, $t/q = \sigma(v)$, $u/q = \sigma(w)$ for some rule $v \to w$ in R. Using Proposition 8 we get: $\lceil t \rceil_{\xi_s(t)(\beta)} = C'[\sigma'(v)]$ and $\lceil u \rceil_{\xi_s(u)(\beta)} = C'[\sigma'(w)]$ for some context C' and some substitution σ'.

So in particular: $\lceil t \rceil_{\xi_s(t)(\beta)} = C'[\sigma'(v)] \to_R C'[\sigma'(w)] = \lceil u \rceil_{\xi_s(u)(\beta)}$.

The reason for the second property is that $s(\beta) \Leftarrow s(\gamma)$; consequently, fewer (but not more) principal subterms might be set to l; again, all these can be reduced to r — this might require ω steps, if the top layer of t is infinite. □

Given an open reduction sequence $f : \alpha \to Ter^\infty(\Sigma)$ and a predicate sequence s we write $\lceil f \rceil_s : \alpha \to Ter^\infty(\Sigma)$ for the function defined as follows (here n is a finite ordinal and λ a limit ordinal or 0):

$$\lceil f \rceil_s(\lambda + 2 \cdot n) = \lceil f(\lambda + n) \rceil_{\xi_s(f(\lambda+n))(\lambda+n)}$$
$$\lceil f \rceil_s(\lambda + 2 \cdot n + 1) = \lceil f(\lambda + n) \rceil_{\xi_s(f(\lambda+n))(\lambda+n+1)}$$

This definition may require some explanation

1. Because f is open, α is a limit ordinal and thus $\lambda + n < \alpha$ implies $\lambda + n + n + 1 < \alpha$, for finite n. Hence α remains as the domain for $\lceil f \rceil_s$.
2. That the case distinction into even and odd ordinals is well-defined and covers all cases follows from the normal-form theorem for ordinals [9, page 323].
3. The purpose of this definition is the following. A single reduction step $f(\beta) \to f(\beta + 1)$ of the original sequence is split into two stages:

   $$\lceil f(\beta) \rceil_{\xi_s(f(\beta))(\beta)} \twoheadrightarrow_w \lceil f(\beta) \rceil_{\xi_s(f(\beta))(\beta+1)} \to_R^= \lceil f(\beta + 1) \rceil_{\xi_s(f(\beta+1))(\beta+1)}$$

   This follows from Proposition 9. Thus the function $\lceil f \rceil_s$ is a weak reduction sequence.
4. As all terms in $\lceil f \rceil_s$ belong to system R the sequence must be convergent — the same argument also explains why $\lceil f \rceil_s$ must be continuous at limit ordinals $\lambda < \alpha$, i.e. why it is a proper sequence.

Lemma 5. *Let $f : \alpha \to Ter^\infty(\Sigma)$ be an open reduction sequence and let R be convergent. Let $p \in \mathsf{PPos}(f)$. Then the sequence f/p is focussed.*

Proof. First notice that $\lceil f \rceil_{f_p}$ converges. By construction, that sequence has an l in position p for all even ordinals (including limits), but it reduces that l to r whenever the corresponding subterm in f fails to have any more reducts in position p. This cannot happen arbitrarily close to α, so there is a cut-off point β after which all $f(\gamma)/p$ have reducts $f(\kappa)/p$, for some $\kappa > \gamma$.

Now assume, for some $\zeta > \beta$ there was no cut-off point ϕ such that for all $\psi \geq \phi$, $f(\zeta)/p \twoheadrightarrow_w f(\psi)/p$. Then we could construct the weak reduction sequence $\lceil f \rceil_{k_{f(\zeta)/p}}$ in R. This replaces all weak reducts of $f(\zeta)/p$ in principal positions by r and all other principal subterms by l. By the assumption and the previous observation we have both l and r occurring at p arbitrarily close to α which contradicts convergence. Hence the assumption was wrong and f must be focussed. □

This result means together with the previous propositions about focussed sequences that if reductions in principal subterm positions would converge then so would the reduction as a whole. But why would they converge? We need an argument that ensures that we can reason inductively.

Lemma 6. *If the combined system has a divergent reduction sequence then it also has a divergent reduction sequence in which all terms have finite rank.*

Proof. Let $f : \alpha \to Ter^{\infty}(\Sigma)$ be that divergent reduction sequence. By Proposition 6 there is a position which changes infinitely often, and we assume p to be one such of minimal length. Clearly, this position is always within the first $|p|$ ranks for all terms of the sequence. We can form $f[|p| + 1 \searrow x]$ for some fixed variable x which by Lemma 1 is also a reduction sequence. Moreover, $f(\beta)$ and $f[|p| + 1 \searrow x](\beta)$ have the same function symbols up to rank $|p|$. Thus the same position p still exhibits infinitely many changes of function symbols in $f[|p| + 1 \searrow x]$, and all terms in that sequence have maximum rank $|p|$. □

The lemma does not hold in the presence of collapsing rules. For example, if $F(x) \to x$ and $G(x) \to x$ are the two systems R and S then we already know that this is not convergent. However, that combination has no divergent reduction of terms with finite rank. What breaks the proof in the presence of collapsing rules is the (lack of) commutation of rewrite steps with principal subterms replacement at rank $|p| + 1$.

The purpose of the lemma is that we can now argue about non-converging reductions by induction on the rank.

Theorem 1. *If two non-collapsing iTRSs R and and S are convergent then so is their disjoint union.*

Proof. By Lemma 6 this can be proved by induction on the minimum required rank for the initial term. The base case is trivial.

Let $f : \alpha \to Ter^{\infty}(\Sigma)$ be an open reduction sequence. If it diverges then by Proposition 6 one position q must witness the divergence infinitely often, and by Lemma 2 this must be below some principal position p of the limit of the top layer. By Lemma 5 the subterm sequence in position p is focussed.

By the induction hypothesis, reductions at lower rank are convergent. Thus Proposition 5 applies and the subterm sequence in position p must converge, contradicting the earlier assumption about q. Thus f converges itself. □

7 Modularity of Strong Convergence

Simonsen showed [10] that *top-termination* (and thus strong convergence) is a modular property of left-linear iTRSs. Left-linearity is indeed an unnecessary constraint:

Theorem 2. *The disjoint union of two strongly converging iTRSs is a strongly converging iTRS.*

Proof. Strong convergence implies that the contributing iTRSs are non-collapsing, hence the previous results about convergence apply. According to [13] an iTRS is strongly converging if and only if it contains no reduction sequence that reduces a root redex infinitely often. Suppose f was such a sequence. But $f[1 \searrow x]$ resides in the iTRS R of the top layer; as the construction preserves all top layer reductions it also preserves all root reductions which contradicts the premise that R is strongly converging. □

8 Conclusion

It has been shown that strong convergence is a modular property of iTRSs, and weak convergence a modular property of non-collapsing iTRSs. Otherwise, the iTRSs involved can have various usually undesirable properties: non-left-linear rules, infinite left-hand sides, and infinitely many rules are all permitted.

The proof for strong convergence is perfectly straightforward — of the results about weak convergence it really only relied upon Lemmas 1 and 2. The proof for weak convergence is comparatively complex.

At the heart of that is a consideration that the reductions in the top layer are limited in the way they can rearrange the principal subterms, in particular that the principal subterms in a fixed position must form a *focussed* sequence. Moreover, abstract reduction systems are converging iff their focussed sequences do, and both observations together give the result.

This is apparently the first non-trivial positive result about weak convergence in the literature. Perhaps even more importantly, the proof of the result offers new tools to deal with weak convergence directly.

References

1. Arnold, A., Nivat, M.: Metric interpretations of infinite trees and semantics of nondeterministic recursive programs. Theoretical Computer Science 11(2), 181–205 (1980)
2. Dershowitz, N., Kaplan, S.: Rewrite, Rewrite, Rewrite, Rewrite, Rewrite,.. In: Principles of Programming Languages, pp. 250–259. ACM, New York (1989)
3. Hajnal, A., Hamburger, P.: Set Theory. London Mathematical Society (1999)
4. Kahrs, S.: Infinitary rewriting: Meta-theory and convergence. Acta Informatica 44(2), 91–121 (2007)
5. Kennaway, R.: On transfinite abstract reduction systems. Technical Report CS-R9205, Centrum voor Wiskunde en Informatica, Amsterdam (1992)
6. Kennaway, R., de Vries, F.-J.: Infinitary Rewriting. In: Term Rewriting Systems, pp. 668–711. Cambridge University Press, Cambridge (2003)
7. Ketema, J.: Böhm-Like Trees for Rewriting. PhD thesis, Vrije Universiteit Amsterdam (2006)
8. Potter, M.D.: Sets: An Introduction. Oxford University Press, Oxford (1990)
9. Sierpiński, W.: Cardinal and Ordinal Numbers. Polish Scientific Publishers (1965)
10. Simonsen, J.G.: On modularity in infinitary term rewriting. Information and Computation 204(6), 957–988 (2006)
11. Terese (ed.): Term Rewriting Systems. Cambridge University Press, Cambridge (2003)
12. Toyama, Y.: On the Chuch-Rosser property for the direct sum of term rewriting systems. Journal of the ACM 34(1), 128–143 (1987)
13. Zantema, H.: Normalisation of infinite terms. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 441–455. Springer, Heidelberg (2008)

A Heterogeneous Pushout Approach to Term-Graph Transformation*

Dominique Duval[1], Rachid Echahed[2], and Frédéric Prost[2]

[1] LJK, Université de Grenoble, France
Dominique.Duval@imag.fr
[2] LIG, CNRS, Université de Grenoble
B.P.53, F-38041 Grenoble, France
Rachid.Echahed@imag.fr, Frederic.Prost@imag.fr

Abstract. We address the problem of cyclic termgraph rewriting. We propose a new framework where rewrite rules are tuples of the form (L, R, τ, σ) such that L and R are termgraphs representing the left-hand and the right-hand sides of the rule, τ is a mapping from the nodes of L to those of R and σ is a partial function from nodes of R to nodes of L. The mapping τ describes how incident edges of the nodes in L are connected in R, it is not required to be a graph morphism as in classical algebraic approaches of graph transformation. The role of σ is to indicate the parts of L to be cloned (copied). Furthermore, we introduce a notion of *heterogeneous pushout* and define rewrite steps as heterogeneous pushouts in a given category. Among the features of the proposed rewrite systems, we quote the ability to perform local and global redirection of pointers, addition and deletion of nodes as well as cloning and collapsing substructures.

1 Introduction

Complex data-structures built by means of records and pointers, can formally be represented by *termgraphs* [2,15,18]. Roughly speaking, a termgraph is a first-order term with possible sharing and cycles. The unravelling of a termgraph is a rational term. Termgraph rewrite systems constitute a high-level framework which allows one to describe, at a very abstract level, algorithms over data-structures with pointers. Thus avoiding, on the one hand, the cumbersome encodings which are needed to translate graphs (data-structures) into trees in the case of programming with first-order term rewrite systems and, on the other hand, the many classical errors which may occur in imperative languages when programming with pointers.

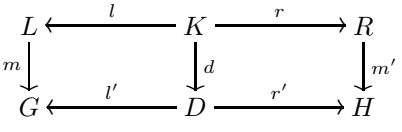
Transforming a termgraph is not an easy task in general. Many different approaches have been proposed in the literature which tackle the problem of termgraph transformation. The algorithmic approach such as [2,8] defines in detail every step involved in the transformation of a termgraph by providing the

* This work has been partly funded by the project ARROWS of the French *Agence Nationale de la Recherche.*

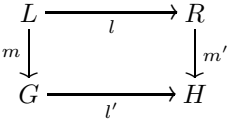
R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 194–208, 2009.
© Springer-Verlag Berlin Heidelberg 2009

corresponding algorithm; for our purpose, this approach is too close to implementation techniques. In [1], equational definition of termgraphs is exploited to define termgraph transformation. These transformations are obtained up to bisimilar structures (two termgraphs are bisimilar if they represent the same rational term). Unfortunately, bisimilarity is not a congruence in general, e.g., the lengths of two bisimilar but different circular lists are not bisimilar.

A more abstract approach to graph transformation is the algebraic one, first proposed in the seminal paper [10]. It defines a rewrite step using the notion of pushouts. The algebraic approach is quite declarative. The details of graph transformations are hidden thanks to pushout constructs. There are mainly two different algebraic approaches, namely the double pushout (DPO) and the single pushout (SPO) approaches, which can be illustrated as follows:



Double pushout: a rewrite step



Single pushout: a rewrite step

In the DPO approach [10,5], a rule is defined as a span, i.e., as a pair of graph morphisms $L \leftarrow K \rightarrow R$. A graph G rewrites into a graph H if and only if there exists a morphism (a matching) $m : L \rightarrow G$, a graph D and graph morphisms d, m', l', r' such that the left and the right squares in the diagram above for a DPO step are pushouts. In general, D is not unique, and sufficient conditions may be given in order to ensure its existence, such as dangling and identification conditions. Since graph morphisms are completely defined, the DPO approach is easy to grasp, but in general this approach fails to specify rules with deletion of nodes, as witnessed by the following example. Let us consider the reduction of the term $f(a)$ by means of the rule $f(x) \rightarrow f(b)$. This rule can be translated into a span $f(x) \leftarrow K \rightarrow f(b)$ for some graph K. When applied to $f(a)$, because of the pushout properties, the constant a must appear in D, hence in H, although $f(b)$ is the only desired result for H, in the context of term rewriting.

In the SPO approach [17,12,13,9], a rule is a *partial* graph morphism $L \rightarrow R$. When a (total) graph morphism $m : L \rightarrow G$ exists, G rewrites to H if and only if the square in the diagram above for a SPO step is a pushout. This approach is appropriate to specify deletion of nodes thanks to partial morphisms. However, in the case of termgraphs, some care should be taken when a node is deleted. Indeed, deletion of a node causes automatically the deletion of its incident edges. This is not sound in the case of termgraphs since each function symbol should have as many successors as its arity.

In this paper, we investigate a new approach to the definition of rewrite steps for cyclic termgraphs. We are interested in rewrite steps such that H is obtained from G by performing one of the six following kinds of actions: (i) addition of new nodes, (ii) redirection of particular edges, (iii) redirection of all incoming edges of a particular node, (iv) deletion of nodes, (v) cloning of nodes, and (vi) collapsing of nodes. In order to deal with these features in a single framework,

we propose a new algebraic approach to define such rewrite steps. Our approach departs from the SPO and the DPO approaches. A rewrite rule is defined as a tuple (L, R, τ, σ) such that L and R are termgraphs, respectively the left-hand side and the right-hand side of the rule, τ is a mapping from the nodes of L into the nodes of R (τ needs not be a graph morphism) and σ is a partial function from the nodes of R into the nodes of L. Roughly speaking, $\tau(p) = n$ indicates that incoming edges of p are to be redirected towards n and $\sigma(n) = p$ indicates that node n should be instantiated as p (parameter passing). We show that whenever a matching $m : L \to G$ exists, then the termgraph G rewrites into a termgraph H. We define the termgraph H as an initial object of a given category, generalizing the definition of a pushout. We call it a *heterogeneous pushout*.

The paper is organized as follows. In section 2 we introduce the basic definitions of graphs and morphisms considered in the paper. In section 3, we give the definition of rewriting, and we illustrate our approach through several examples in section 4. A comparison with related work is done in section 5, and concluding remarks are given in section 6.

2 Graphs

In this section are given some basic definitions. We assume the reader is familiar with category theory. The missing definitions may be consulted in [14].

Throughout this paper, a signature Ω is fixed. Each operation symbol $\omega \in \Omega$ is endowed with an *arity* $\mathrm{ar}(\omega) \in \mathbb{N}$. For each set X, the set of strings over X is denoted X^*, and for each function $f : X \to Y$, the function $f^* : X^* \to Y^*$ is defined by $f^*(x_1 \dots x_n) = f(x_1) \dots f(x_n)$. In addition, we denote by **Set** the category of sets.

Definition 1 (Graph). *A* termgraph*, or simply a* graph $G = (\mathcal{N}, \mathcal{D}, \mathcal{L}, \mathcal{S})$ *is made of a set of* nodes $\mathcal{N}$ *and a subset of* labeled nodes $\mathcal{D} \subseteq \mathcal{N}$*, which is the domain for a* labeling function $\mathcal{L} : \mathcal{D} \to \Omega$ *and a* successor function $\mathcal{S} : \mathcal{D} \to \mathcal{N}^*$*, such that for each labeled node n, the length of the string $\mathcal{S}(n)$ is the arity of the operation $\mathcal{L}(n)$. For each labeled node n the fact that $\omega = \mathcal{L}(n)$ is written $n{:}\omega$, and each unlabeled node n may be written as $n{:}\bullet$, so that the symbol $\bullet$ is a kind of anonymous variable. A* graph homomorphism*, or simply a* graph morphism $g : G \to H$*, where $G = (\mathcal{N}_G, \mathcal{D}_G, \mathcal{L}_G, \mathcal{S}_G)$ and $H = (\mathcal{N}_H, \mathcal{D}_H, \mathcal{L}_H, \mathcal{S}_H)$ are graphs, is a function $g : \mathcal{N}_G \to \mathcal{N}_H$ which preserves the labeled nodes and the labeling and successor functions. This means that $g(\mathcal{D}_G) \subseteq \mathcal{D}_H$, and for each labeled node n, $\mathcal{L}_H(g(n)) = \mathcal{L}_G(n)$ and $\mathcal{S}_H(g(n)) = g^*(\mathcal{S}_G(n))$ (the image of an unlabeled node may be any node). This yields the category* **Gr** *of graphs.*

Definition 2 (Node functor). *The* node functor $|-| : \mathbf{Gr} \to \mathbf{Set}$ *maps each graph $G = (\mathcal{N}, \mathcal{D}, \mathcal{L}, \mathcal{S})$ to its set of nodes $|G| = \mathcal{N}$ and each graph morphism $g : G \to H$ to its underlying function $|g| : |G| \to |H|$. In this paper, it is also called the* underlying *functor.*

We may denote g instead of $|g|$ since the node functor is faithful, which means that a graph morphism is determined by its underlying function on nodes. The

faithfulness of the node functor implies that a diagram of graphs is commutative if and only if its underlying diagram of sets is commutative.

The *graphic functions* and the *strictly graphic functions*, as defined now, can be seen as "weak" graph morphisms. They will be used in section 3 to relate graphs involved in a rewrite step.

Definition 3 (Graphic functions). *Let G and H be graphs and $\gamma : |G| \to |H|$ a function. For each node n of G, γ is* graphic at n *if either n is unlabeled or both n and $\gamma(n)$ are labeled, $\mathcal{L}_H(\gamma(n)) = \mathcal{L}_G(n)$ and $\mathcal{S}_H(\gamma(n)) = \gamma^*(\mathcal{S}_G(n))$. And γ is* strictly graphic at n *if either both n and $\gamma(n)$ are unlabeled or both n and $\gamma(n)$ are labeled, $\mathcal{L}_H(\gamma(n)) = \mathcal{L}_G(n)$ and $\mathcal{S}_H(\gamma(n)) = \gamma^*(\mathcal{S}_G(n))$. For each set of nodes Γ of G, γ is* graphic (resp. strictly graphic) on Γ *if γ is graphic (resp. strictly graphic) at every node in Γ.*

Example 1. Let us consider the following graphs G_1 and G_2:

Let $\gamma : |G_1| \to |G_2|$ be the function defined by $\gamma = \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c, 4 \mapsto d\}$. It is easy to check that γ is strictly graphic on $\{1, 3\}$, is graphic but not strictly graphic on $\{1, 2, 3\}$, and is not graphic on $\{1, 2, 3, 4\}$.

It should be noted that the property of being graphic (resp. strictly graphic) on Γ involves the successors of the nodes in Γ, which may be outside Γ. In addition, it is clear that a function $\gamma : |G| \to |H|$ underlies a graph morphism $g : G \to H$ if and only if it is graphic on $|G|$. The next straightforward result will be useful.

Lemma 1. *Let G, H, H' be graphs and let $\gamma : |G| \to |H|$, $\gamma' : |G| \to |H'|$, $\eta : |H| \to |H'|$ be functions such that $\gamma' = \eta \circ \gamma$. Let Γ be a set of nodes of G. If γ is strictly graphic on Γ and γ' is graphic on Γ, then η is graphic on $\gamma(\Gamma)$.*

3 Rewriting

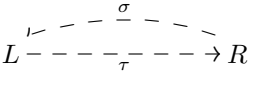
Roughly speaking, in the context of graph rewriting, a rewrite rule has a left-hand side graph L and a right-hand side graph R, and a rewrite step applied to a graph G with an occurrence of L returns a graph H with an occurrence of R, by replacing L by R in G. The meaning of this "replacement" is quite clear for the nodes, as well as for the edges of G that connect two nodes outside L. When a labeled node p in G outside L has its i-th successor p' inside L, then p must have some i-th successor n' in H. For this purpose, we introduce a function τ (τ for "target") from the nodes of L to the nodes of R, and we decide that n' must be $\tau(p')$. On the other hand, when a labeled node p in G inside L has an i-th successor p', then we may require that some node n' in H has the same label as p and has as its i-th successor either p', if it is outside L, or $\tau(p')$ otherwise; then

n' is called a τ-clone of p. Since each node in L may have an arbitrary number of clones (maybe no clone at all), and a node in R cannot be a clone of more than one node in L, this is specified thanks to a partial function σ (σ for "source") from the nodes of R to the nodes of L, which maps the clones of p to p. Partial functions are denoted with the symbol "$\rightharpoonup$", the domain of a partial function σ is denoted $\mathrm{Dom}(\sigma)$, and the composition of partial functions is defined as usual. The main result is theorem 1: under relevant definitions and assumptions, for each rewrite rule T and matching m there is a *heterogeneous pushout* of T and m, which can be built explicitly from a pushout of sets.

Definition 4 (Clones). *Let G and H be graphs and $\tau : |G| \to |H|$ a function. Then $p \in |H|$ is a* τ-clone *of $q \in |G|$ when: p is labeled if and only if q is labeled, and then $\mathcal{L}_H(p) = \mathcal{L}_G(q)$ and $\mathcal{S}_H(p) = \tau^*(\mathcal{S}_G(q))$. It is* not *required that $p = \tau(q)$.*

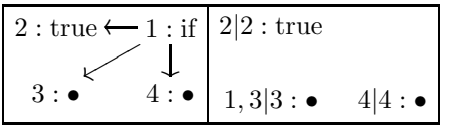
Definition 5 (Rewrite rule). *A* rewrite rule *is a tuple (L, R, τ, σ) made of two graphs L and R, a function $\tau : |L| \to |R|$ and a partial function $\sigma : |R| \rightharpoonup |L|$ such that each node n in the domain of σ is unlabeled or is a τ-clone of $\sigma(n)$. A* morphism of rewrite rules, *from $T = (L, R, \tau, \sigma)$ to $T_1 = (L_1, R_1, \tau_1, \sigma_1)$ is a pair of graph morphisms (m, d) with $m : L \to L_1$ and $d : R \to R_1$ such that $|d| \circ \tau = \tau_1 \circ |m|$, $d(\mathrm{Dom}(\sigma)) \subseteq \mathrm{Dom}(\sigma_1)$ and $|m| \circ \sigma = \sigma_1 \circ |d|$ on $\mathrm{Dom}(\sigma)$.*

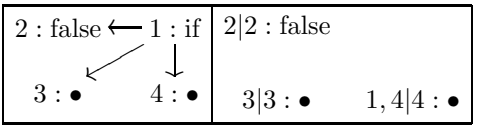
In this paper, the illustrations take place either in the category **Set** of sets or in a heterogeneous framework where the points stand for graphs, the solid arrows for graph morphisms and the dashed arrows for functions on nodes. So, a rewrite rule $T = (L, R, \tau, \sigma)$ will be illustrated as:



In order to ease the reading of the examples, a rule is depicted as $\boxed{L|R}$. In addition, each node n in R in the image of τ is named $n = x, y, \ldots | w$ where $x, y, \ldots$ are the names of the nodes in L such that $\tau(x) = \tau(y) = \ldots = n$ and where $\sigma(n) = w$. Whenever n is not in the image of τ then it is named $n = x|w$ where the name x is new (i.e., x does not appear in L) and where $\sigma(n) = w$. In both cases, the "$|w$" part is omitted when n is not in the domain of σ.

Example 2 (if-then-else). The following rewrite rules define the "if-then-else" operator as it behaves in classical imperative languages. We assume that the three arguments of an "if-then-else" expression may be shared.



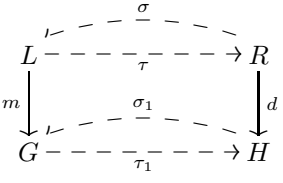


The definition of τ ensures that the "if-then-else" expression is replaced by its value $\tau(3) = 1, 3|3$ (resp. $\tau(4) = 1, 4|4$). The definition of σ indicates that

the value of the "if-then-else" is its second (resp. third) argument specified by $\sigma(1,3|3) = 3$ (resp. $\sigma(1,4|4) = 4$). Notice that if σ were defined as the empty function, the "if-then-else" expression would evaluate to an unlabeled node. In addition, in both rules, we have $\sigma(2|2) = 2$ which ensures, in case node 2 is shared, that its incident edges remain unchanged after a rewrite step. Finally, the reader may verify that these two rules are sufficient to handle all possible cases, even when the arguments of an "if-then-else" expression are shared, thanks to the conditions on matching substitutions given later in definition 8.

It can be noted that each graph morphism $t : L \to R$ determines a rewrite rule where $\tau = |t|$ and σ is defined nowhere. In this case, for each graph morphism $m : L \to G$ the pushout of t and m in the category **Gr**, when it exists, is the initial object in the category of cones over t and m. Let us adapt this definition to any rewrite rule $T = (L, R, \tau, \sigma)$ and any graph morphism $m : L \to G$.

Definition 6 (Heterogeneous cone). *Let $T = (L, R, \tau, \sigma)$ be a rewrite rule and $m : L \to G$ a graph morphism. A* heterogeneous cone over T and m *is a tuple (H, τ_1, d, σ_1) made of a graph H, a function $\tau_1 : |G| \to |H|$, a graph morphism $d : R \to H$ and a partial function $\sigma_1 : |H| \rightharpoonup |G|$ such that $T_1 = (G, H, \tau_1, \sigma_1)$ is a rewrite rule, $(m, d) : T \to T_1$ is a morphism of rewrite rules, τ_1 is graphic on $|G| - |m(L)|$ and n_1 is a τ_1-clone of $\sigma_1(n_1)$ for each n_1 in the domain of σ_1.*



A morphism of heterogeneous cones over T and m, *say $h : (H, \tau_1, d, \sigma_1) \to (H', \tau'_1, d', \sigma'_1)$, is a graph morphism $h : H \to H'$ such that $|h| \circ \tau_1 = \tau'_1$, $h \circ d = d'$, $h(\mathrm{Dom}(\sigma_1)) \subseteq \mathrm{Dom}(\sigma'_1)$ and $\sigma'_1 \circ |h| = \sigma_1$ on $\mathrm{Dom}(\sigma_1)$.*

This yields the category $\mathbf{C}_{T,m}$ of heterogeneous cones over T and m.

Definition 7 (Heterogeneous pushout). *Let $T = (L, R, \tau, \sigma)$ be a rewrite rule and $m : L \to G$ a graph morphism. A* heterogeneous pushout of T and m *is an initial object in the category $\mathbf{C}_{T,m}$ of heterogeneous cones over T and m.*

When a heterogeneous pushout exists, its initiality implies that it is unique up to an isomorphism of heterogeneous cones. In theorem 1 we prove the existence of a heterogeneous pushout of T and m under some injectivity assumption on m.

Definition 8 (Matching). *A* matching *with respect to a rewrite rule $T = (L, R, \tau, \sigma)$ is a graph morphism $m : L \to G$ such that if $m(p) = m(p')$ for distinct nodes p and p' in L then $\tau(p)$ and $\tau(p')$ are in $\mathrm{Dom}(\sigma)$ and $m(\sigma(\tau(p))) = m(\sigma(\tau(p')))$ in G.*

Proposition 1. *Let $T = (L, R, \tau, \sigma)$ be a rewrite rule and $m : L \to G$ a matching with respect to T. Then the pushout of τ and $|m|$ in* **Set***:*

$$\begin{array}{ccc} |L| & \xrightarrow{\tau} & |R| \\ {\scriptstyle |m|}\downarrow & & \downarrow{\scriptstyle \delta} \\ |G| & \xrightarrow[\tau_1]{} & \mathcal{H} \end{array}$$

satisfies $\mathcal{H} = \tau_1(\Gamma) + \delta(\Delta) + \delta(\Sigma)$ where $\Gamma = |G| - |m(L)|$, $\Sigma = \mathrm{Dom}(\sigma)$, $\Delta = |R| - \Sigma$ and: the restriction of $\tau_1 : \Gamma \to \tau_1(\Gamma)$ is bijective, the restriction of $\delta : \Delta \to \delta(\Delta)$ is bijective, and the restriction of $\delta : \Sigma \to \delta(\Sigma)$ is such that if $\delta(n) = \delta(n')$ for distinct nodes n and n' in Σ then $m(\sigma(n)) = m(\sigma(n'))$ in G. In addition, there is a unique partial function $\sigma_1 : \mathcal{H} \rightharpoonup |G|$ with domain $\delta(\Sigma)$ such that $|m| \circ \sigma = \sigma_1 \circ \delta$.

Proof. Clearly $\mathcal{H} = \tau_1(\Gamma) + \delta(|R|)$ and the restriction of $\tau_1 : \Gamma \to \tau_1(\Gamma)$ is bijective. If $\delta(n) = \delta(n')$ for distinct nodes n and n' in R, then there is a chain from n to n' made of pieces like this one:

$$\widetilde{n} \xleftarrow{\ \tau\ } p \xmapsto{\ |m|\ } p_1 \xleftarrow{\ |m|\ } p' \xmapsto{\ \tau\ } \widetilde{n}'$$

with $\widetilde{n}, \widetilde{n}' \in |R|$, $p, p' \in |L|$, $p_1 \in |G|$, and it can be assumed that $\widetilde{n} \neq \widetilde{n}'$ and $p \neq p'$. Since m is a matching, $\widetilde{n}$ and $\widetilde{n}'$ are in Σ and $m(\sigma(\widetilde{n})) = m(\sigma(\widetilde{n}'))$. The decomposition of $\mathcal{H}$ follows. Now, let $n_1 \in \delta(\Sigma)$ and let us choose some $n \in \Sigma$ such that $n_1 = \delta(n)$. If σ_1 exists, then $\sigma_1(n_1) = \sigma_1(\delta(n)) = m(\sigma(n))$. On the other hand, if $n' \in \Sigma$ is another node such that $n_1 = \delta(n)$, then we have just proved that $m(\sigma(n)) = m(\sigma(n'))$, so that $m(\sigma(n))$ does not depend on the choice of n, it depends only on n_1. So, there is a unique $\sigma_1 : \mathcal{H} \rightharpoonup |G|$ as required, it is defined by $\sigma_1(n_1) = m(\sigma(n))$ for any $n \in \Sigma$ such that $n_1 = \delta(n)$.

Proposition 2. *Let $m : L \to G$ be a matching with respect to a rewrite rule $T = (L, R, \tau, \sigma)$. The pushout of τ and $|m|$ in* **Set***, with σ_1 as in proposition 1, underlies a heterogeneous cone over T and m.*

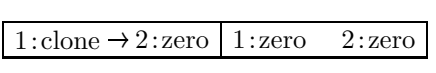
Proof. First, let us define a graph H with set of nodes $\mathcal{H}$. By exploiting proposition 1, and with the same notations, a graph H with set of nodes $\mathcal{H}$ is defined by imposing that τ_1 is strictly graphic on Γ, that δ is strictly graphic on Δ, and that each node $n_1 \in \delta(\Sigma)$ is a τ_1-clone of q_1, where $q_1 = \sigma_1(n_1)$. Now, let us prove that δ underlies a graph morphism $d : R \to H$. Since δ is graphic on Δ, we have to prove that δ is also graphic on Σ. Let $n \in \Sigma$ and $n_1 = \delta(n)$. If n is unlabeled there is nothing to prove, otherwise let $q = \sigma(n)$, then q is labeled, $\mathcal{L}_R(n) = \mathcal{L}_L(q)$ and $\mathcal{S}_R(n) = \tau^*(\mathcal{S}_L(q))$. Then $m(q) = m(\sigma(n)) = \sigma_1(\delta(n)) = q_1$, and from the fact that m is a graph morphism we get $\mathcal{L}_L(q) = \mathcal{L}_G(q_1)$ and $|m|^*(\mathcal{S}_L(q)) = \mathcal{S}_G(q_1)$. The definition of H imposes $\mathcal{L}_G(q_1) = \mathcal{L}_H(n_1)$ and $\tau_1^*(\mathcal{S}_G(q_1)) = \mathcal{S}_H(n_1)$. Altogether, $\mathcal{L}_R(n) = \mathcal{L}_H(n_1)$ and $\mathcal{S}_H(n_1) = (\tau_1^*(|m|^*(\mathcal{S}_G(q))) = \delta^*(\tau^*(\mathcal{S}_G(q))) = \delta^*(\mathcal{S}_R(n))$, so that indeed δ is also graphic on Σ. Finally, it is easy to check that this yields a heterogeneous cone over T and m.

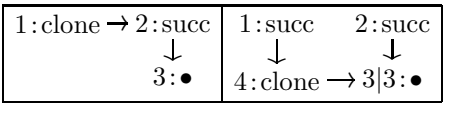
Theorem 1. *Given a rewrite rule $T = (L, R, \tau, \sigma)$ and a matching $m : L \to G$ with respect to T, the heterogeneous cone $(m, d) : T \to T_1$ over T and m defined in proposition 2 is a heterogeneous pushout of T and m.*

Proof. As in proposition 2, let $T_1 = (G, H, \tau_1, \sigma_1)$. Let us consider any heterogeneous cone $(m, d') : T \to T'_1$ over T and m, with $T'_1 = (G', H', \tau'_1, \sigma'_1)$. Since (m, d) underlies a pushout of sets, there is a unique function $\eta : |H| \to |H'|$ such that $\eta \circ |d| = |d'|$ and $\eta \circ \tau_1 = \tau'_1$. Let $\Sigma = \mathrm{Dom}(\sigma)$ and $\Sigma_1 = \mathrm{Dom}(\sigma_1)$. Because the node functor is faithful, the result will follow if we can prove that $\eta(\Sigma_1) \subseteq \Sigma'_1$ and $\sigma'_1 \circ \eta = \sigma_1$ on Σ_1, and that η underlies a graph morphism. We have $\eta(\Sigma_1) = \eta(d(\Sigma)) = d'(\Sigma) \subseteq \Sigma'_1$, and for each $n_1 \in \Sigma_1$, let $n \in \Sigma$ such that $n_1 = d(n)$, then on one hand $\sigma'_1(\eta(n_1)) = \sigma'_1(\eta(d(n))) = \sigma'_1(d'(n)) = m(\sigma(n))$ and on the other hand $\sigma_1(n_1) = \sigma_1(d(n)) = m(\sigma(n))$, hence as required $\sigma'_1(\eta(n_1)) = \sigma_1(n_1)$. In order to check that η underlies a graph morphism $h : H \to H'$, we use the decomposition of $\mathcal{H}$ from proposition 1 and the construction of the heterogeneous cone (m, d) in proposition 2. It follows immediately from lemma 1 that η is graphic on $\tau_1(\Gamma)$ and also on $d(\Delta)$. Let us prove that η is graphic on Σ_1. Let $n_1 \in \Sigma_1$, $q_1 = \sigma_1(n_1)$ and $n'_1 = \eta(n_1)$. Then $q_1 = \sigma'_1(n'_1)$ because $\sigma'_1 \circ \eta = \sigma_1$. So, n_1 is a τ_1-clone of q_1 and n'_1 is a τ'_1-clone of the same node q_1. This means that $\mathcal{L}_{H'}(n'_1) = \mathcal{L}_G(q_1) = \mathcal{L}_H(n_1)$ and that $\mathcal{S}_{H'}(n'_1) = (\tau'_1)^*(\mathcal{S}_G(q_1)) = \eta^*(\tau_1^*(\mathcal{S}_G(q_1))) = \eta^*(n_1)$. So, η is graphic on Σ_1, and since $d(\Sigma) \subseteq \Sigma_1$, it follows that η is graphic on $d(\Sigma)$. Altogether, η is graphic on the whole of $|H|$, which means that $\eta = |h|$ for a graph morphism $h : H \to H'$. This concludes the proof.

Definition 9 (Rewrite step). *Given a rewrite rule $T = (L, R, \tau, \sigma)$ and a matching $m : L \to G$ with respect to T, the corresponding* rewrite step *builds the graph morphism $d : R \to H$, obtained from the heterogeneous pushout of T and m.*

Example 3 (Cloning data-structures). Here are two rules for cloning natural numbers, encoded with succ and zero. These rules can be generalized to any data-structure as presented in section 5.



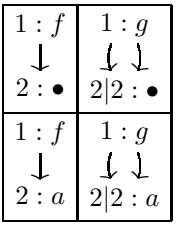


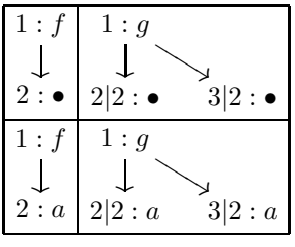
The first rule takes care of the cloning of *zero*. Given a matching $m : L \to G$, since $\tau(1) = 1$ this rule redirects all edges in G with target $m(1 : \mathrm{clone})$ to edges in H with target $d(1 : \mathrm{zero})$, and since $\tau(2) = 2$ the edges in G with target $m(2 : \mathrm{zero})$ remain "unchanged" in H, in the sense that their target is $d(2 : \mathrm{zero})$. The second rule takes care of the cloning of the non-zero naturals. Since $\sigma(3|3) = 3$, the label and successors of the node $m(3)$ in G will be "the same as" the label and successors of the node $d(3|3)$ in H. Notice that, in this case, it would not be allowed to define $\sigma(4) = 2$ because node 4 in R is labeled by clone and node 2 in L is labeled by succ, thus breaking the τ-clone condition.

Let us represent a rewrite step from G to H performed by a rewrite rule (L, R, τ, σ) as in the figure opposite, with the same notations as above regarding node names in the right-hand side of the rule. When the matching, m, is injective on nodes we denote $m(p) = p$.

L	R
G	H

Example 4. Let us consider the "rule" $f(x) \to g(x, x)$. In our framework, this rule may be translated to several different rules according to the way $g(x, x)$ is represented as a termgraph and to the way the functions τ and σ are defined. For instance, here are two different rules and their application to the termgraph $1 : f(2 : a)$, with the matching preserving the names of the nodes. The second rule provides two clones $2|2 : a$ and $3|2 : a$ in H to the node $2 : a$ in G.





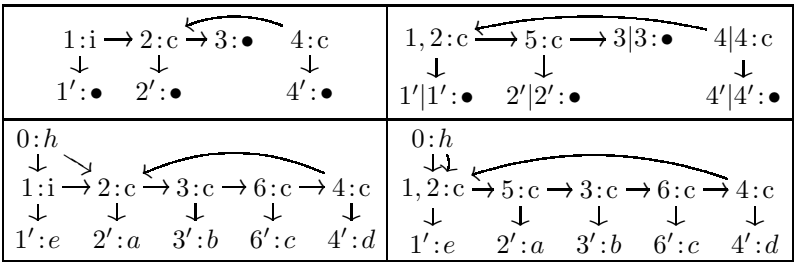
Since the rewriting of termgraphs with heterogeneous pushouts relies on a pushout on the underlying nodes (proposition 1), and also because of the condition on matching (definition 8), there is an obvious relation between the size of the rewritten termgraph and the size of the original termgraph, for each rewrite step. Moreover, this relation can be inferred at the level of rules. So, one can analyze memory usage of a program simply by inspecting its rules. Proposition 3, where $\sharp$ denotes the cardinal, states this formally. Therefore, if the size of a termgraph is considered as the measure of the memory used (thus putting aside unreachability issues), then it is possible to statically compute, for each rule separately, the amount of memory needed, or freed, by a rewriting step.

Proposition 3. *Let $T = (L, R, \tau, \sigma)$ be a rewrite rule, $m : L \to G$ a matching and $d : R \to H$ the result of the rewrite step. Then $\sharp|H| - \sharp|G| = \sharp|R| - \sharp|L|$.*

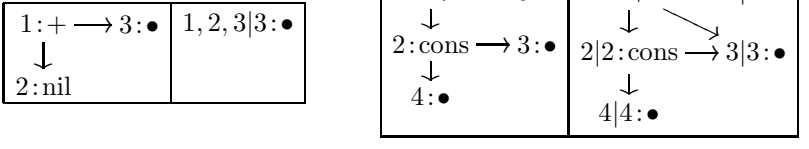
4 Examples

In this section, we provide several examples illustrating our framework. For better readability, when rewriting a termgraph G into H, the description of σ_1 in the graph H will now be omitted.

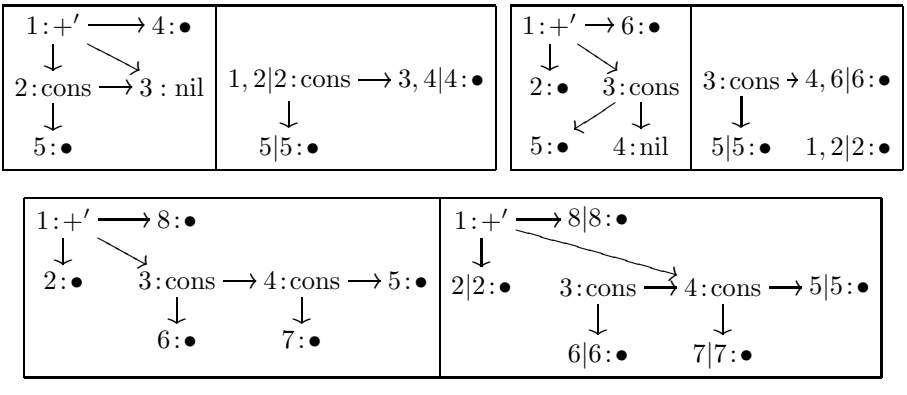
Example 5 (Insertion in a circular list). Here is a rule for the insertion of an element at the head of a circular list of size greater than one. In the left-hand side of this rule, node 2 is the head of the list, and node 4 is the last element of the list. The pointer from node 4 to the head of the list is moved from 2 (in L) to a new node $1, 2$ (in R). The definition of τ is such that all pointers to the head of the list are moved from 2 to $1, 2$. We apply the rule on a circular list of four items. We note c and i for cons and insert, respectively.



Example 6 (Appending linked lists). We now consider two rules for the operation "+" which appends, in place, two linked lists. The lists are built with the constructors cons and nil.



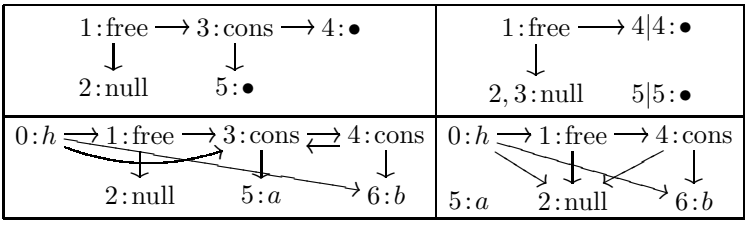
The first rule above takes care of the base case, when the first argument is nil. The second rule says that when the first argument of + is a non-empty list, then an auxiliary function "+′" of arity 3 is called. The role of this function is to go through the first list until its end and to concatenate the two lists by pointer redirection. The following three rules define the operation +′.



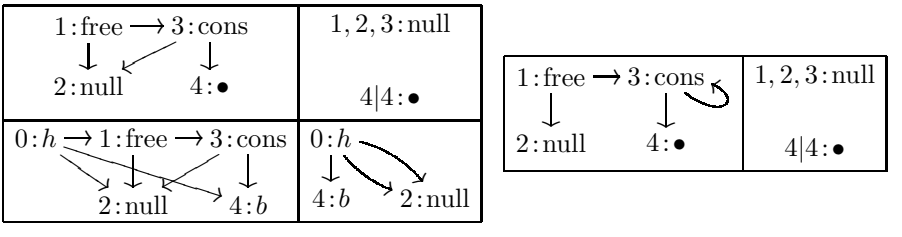
The first rule considers the case when the first list consists of one element. The second rule defines the case when the last element of the first list is reached. In this case, the edge $3 \to 4$ in L is redirected as $3 \to 4, 6|6$ in R, which is the head of the second list to append. The overall result of the operation +′ is the node $\tau(1) = 1, 2|2$, which is the head of first list. The third rule performs the traversal of the first list.

Example 7 (Memory freeing). In this example we show how we can free the memory used by the "cons" nodes of a circular list. As we are concerned with

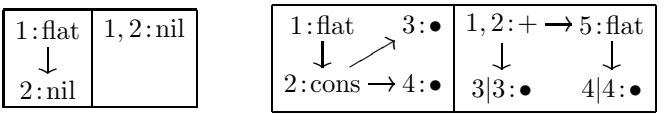
termgraphs where every function symbol has a fixed arity, it is not allowed to create dangling pointers nor to remove useless pointers. This constraint is expressed by the fact that every node in a left-hand side L must have an image in the right-hand side R by τ. The operation free has two arguments. The first one is a particular node labeled by a constant null, it is dedicated to be the target of the edges which were pointing to the freed nodes. The second argument of free is the list of cells to be freed. Here is a rule for defining the operation free in the case of a list with at least two elements. We illustrate its application on a list of length two. Notice that pointers incoming to nodes 3 and 5 are redirected towards node 2 in H.



For lists with one element, there are two cases to be considered. The first rule specifies the case where the last element of the list is obtained after freeing other elements of the list. We illustrate the rewrite step on the graph obtained earlier (up to renaming of nodes). The second rule deals with the special case of circular lists of size one.



Example 8 (Memory usage analysis). Predicting memory usage of an algorithm can be of great interest for many applications, especially those involved within embedded systems. There exist several methods to analyse memory usage. For example, in [11], a very powerful method based on a type system enriched with resource annotations has been proposed. The authors succeeded in analysing several examples, but failed to tackle a program which flattens a list of lists. As an application of proposition 3, we show how our framework can be used to analyse the memory usage of such an algorithm. The program which flattens a list of lists, consists of the following rules, the first one is for the base case (the empty list) and the second rule deals with a non-empty list of lists. Here $+$ stands for the append operator and $+'$ for its auxiliary operator, as in example 6.



Thus, memory usage by flattening a list can be analyzed considering seven rules: two for flat, two for + and three for $+'$. Inspection of the recursion rules for the three functions shows that the number of nodes is unchanged. The halt case rule for flat frees one memory cell. The halt case rules, both for + and $+'$, free two memory cells. Now, simple reasoning on the rules involved in the evaluation of flat shows that flat(ℓ) frees exactly $2|\ell| + 1$ memory cells, where $|\ell|$ is the size of the list ℓ. Indeed, there is one halt case for flat, and for each element of ℓ there is one halt case for + or another one for $+'$.

5 Related Work

Cloning is also one of the features of the sesqui-pushout approach to graph transformation [4]. In this approach, a rule is a span $L \leftarrow K \rightarrow R$ and the application of a rule to a graph G can be illustrated by the same figure as for a DPO step (as in the introduction), where the right-hand side is a pushout as in the DPO approach but the left-hand side is a pullback, and moreover it is a final pullback complement. The graphs considered in [4] are defined as $G = (V, E, \mathtt{src} : E \rightarrow V, \mathtt{tgt} : E \rightarrow V)$ where V and E are the sets of vertices and edges respectively, and the connections of edges are defined by the functions `src` (source) and `tgt` (target). Nodes are not endowed with arities and thus they may have an arbitrary amount of outgoing edges. This fact, together with the use of final pullback complements, makes the sesqui-pushout approach different from our framework. We illustrate this difference through the cloning of nodes. According to the sesqui-pushout approach, the cloning of a node is, roughly speaking, performed by copying a node together with all its incident edges (incoming and outgoing edges). In our framework, a node is copied only with its outgoing edges. Let us consider for instance the termgraph $h(f(2:a), 2)$ in which the subgraph $f(2:a)$ is supposed to be transformed into $g(2, 3)$ where nodes 2 and 3 are clones of $2:a$. Then according to our framework, this transformation can be achieved by means of the following rule. The application of this rule to $h(f(2:a), 2)$ yields the graph $h(g(2:a, 3:a), 2)$.

$$\boxed{2:\bullet \leftarrow 1:f} \quad \boxed{2|2:\bullet \leftarrow 1:g \rightarrow 3|2:\bullet}$$

Using the sesqui-pushout approach, we get a rule of the following shape.

$$\boxed{2:\bullet \leftarrow 1:f} \longleftarrow \boxed{K} \longrightarrow \boxed{2:\bullet \leftarrow 1:g \rightarrow 3:\bullet}$$

Indeed, K should encode the cloning of the instance of node 2 as well as the replacement of f by g, thus K should include at least three unlabeled nodes. The application of this rule to $h(f(2:a), 2)$ yields the graph $h(g(2:a, 3:a), 2, 3)$, where the operation h has three arguments, because each of the clones $2:a$ and $3:a$ requires the cloning of all incoming edges.

Cloning has also been subject of interest in [6]. The authors considered rewrite rules of the form $S := R$ where S is a star, i.e., S is a (nonterminal) node surrounded by its adjacent nodes together with the edges that connect them.

Rewrite rules which perform the cloning of a node have been given in [6, Def. 6]. These rules show how a star can be removed, kept identical to itself or copied (cloned) more than once. Here again, unlike our framework, the cloning does not care about the arity of the nodes and, as in the case of the sesqui-pushout approach, a node is copied together with all its incoming and outgoing edges. If we consider the termgraph $h(f(2:a),2)$ again and clone twice the node $2:a$, then according to [6] we get the graph $h(f(2:a,3:a),2,3)$ where both h and f have augmented their arity when copying the incoming edges to the clone $3:a$.

A categorical framework dedicated to *cyclic* termgraph transformation can be found in [3] where the authors propose, following [16], a 2-categorical presentation of termgraph rewriting. They almost succeed in representing the full operational view of termgraph rewriting as defined in [2], but they differ on rewriting circular redexes. For example, the application of the rewrite rule $f(x) \to x$ on the termgraph $n : f(n)$ yields the same termgraph (i.e., $n : f(n)$) according to [2] but yields an unlabeled node, say $p:\bullet$, according to [3]. With our definition of rewrite rules, it is possible to encode exactly the algorithmic approach [2] by simply stating that node $n, m|m$ is a clone of node m in the rule:

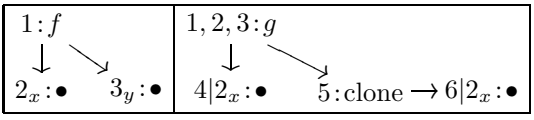
$n:f \longrightarrow m:\bullet$	$n, m\|m:\bullet$

In general, term rewriting and termgraph rewriting do not coincide [15]. However, thanks to its cloning facilities, our framework can simulate term rewriting in the case of left linear term rewrite systems. Indeed, a term rewrite rule $l \to r$ where l is a linear term (i.e., variables in l occur only once in l), can be transformed into a rule (L, R, τ, σ) where $L = \texttt{l2L}(l)$ and $R = \texttt{r2R}(r)$ are termgraphs corresponding respectively to l and r, by the transformations `l2L` and `r2R` defined below. Notice that the transformation of the right-hand side takes into account the cloning of variable instances, this happens when a right-hand side is not linear. The aim of τ, when simulating term rewriting, consists in indicating the replacement of the root of L by that of R, i.e., $\tau(\texttt{root}(L)) = \texttt{root}(R)$. The images via τ of the remaining nodes are not significant, so that $\tau(|L|) = \texttt{root}(R)$ is a possible choice for τ. The function σ indicates the parts that should be cloned, it plays an important role in encoding the use of variables in the right-hand side. Every occurrence of a variable x in r corresponds to a non labeled node $p:\bullet$ in R. In this case, by definition of rewrite rules, x appears in l and thus a corresponding non labeled node $p_x:\bullet$ appears in L, thus we state that p is a clone of p_x by $\sigma(p) = p_x$. Now we define the transformation functions `l2L` and `r2R`.

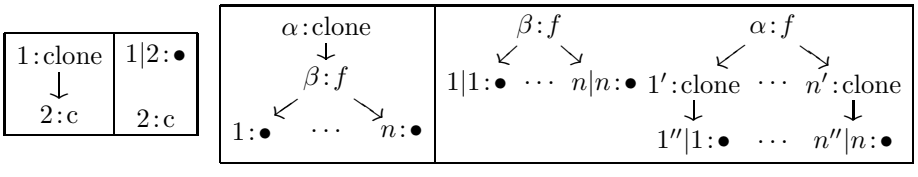
- If c is a constant then $\texttt{l2L}(c) = p:c$ and $\texttt{r2R}(c) = p:c$ where node p is new.
- If f is an operation and the t_i's are terms then $\texttt{l2L}(f(t_1,\ldots,t_n) = p : f(\texttt{l2L}(t_1),\ldots,\texttt{l2L}(t_n))$ and $\texttt{r2R}(f(t_1,\ldots,t_n) = p : f(\texttt{r2R}(t_1),\ldots,\texttt{r2R}(t_n))$ where node p is new.
- If x is a variable then
  - $\texttt{l2L}(x) = p_x:\bullet$ where node p_x is new

- $\texttt{r2R}(x) = p{:}\bullet$ for the first occurrence of x in r where node p is new and $\sigma(p) = p_x$, while $\texttt{r2R}(x) = q{:}\mathrm{clone}(p{:}\bullet)$ otherwise, where nodes p and q are new and $\sigma(p) = p_x$.

For example, the term rewrite rule $f(x, y) \to g(x, x)$ is transformed as:



When applying a matching substitution m over the right-hand side r of a term rewrite rule, for every variable x of the left-hand side m constructs as many copies of the instance $m(x)$ as the number of occurrences of x in r. The aim of $q{:}\mathrm{clone}(p{:}\bullet)$ in the definition of $\texttt{r2R}(x)$ is to mimic the application of a matching substitution on the right-hand side $m(r)$. The function clone builds a copy of its argument, it can be defined for all operators of a given signature as follows, where c is a constant and f is an operation of arity n. We write $\to^*_{\texttt{CLONE}}$ the rewrite relation induced by the following rules.



Proposition 4. *Let $\mathcal{R}$ be a left linear term rewrite system built over a signature Ω. Let $T(\mathcal{R})$ be the termgraph rewrite system made of the rewrite rules which define the function clone over the operation symbols in Ω, together with the transformations of the rules $l \to r$ in R. Let t be a ground term. If $t \to_{\mathcal{R}} t'$ then there exists a termgraph G_1 such that $\texttt{12L}(t) \to_{T(\mathcal{R})} G_1 \to^*_{\texttt{CLONE}} \texttt{12L}(t')$.*

The proof of this proposition is quite obvious, because the term t is assumed to be ground. However, reduction of non linear terms is not allowed by the given transformations. Indeed, variables cannot be cloned by transformation 12L (but only renamed). For example $\texttt{12L}(f(x, x))$ is the termgraph $1 : f(2_x : \bullet, 3_x : \bullet)$, which is not a sound translation of $f(x, x)$. For the instances of nodes 2_x and 3_x are not supposed to be isomorphic.

6 Conclusion

In this paper, we have proposed a new way to define termgraph rewrite rules, and we have defined a rewrite step as a heterogeneous pushout in an appropriate category. The proposed rewrite systems offer the possibility to transform cyclic termgraphs either by performing local edge redirections or global edge redirections, as defined following a DPO approach in [7], and in addition, it provides new features such as cloning or deleting nodes. Future work includes the generalization of the proposed systems to other graphs less constrained than termgraphs.

Acknowledgements

We would like to thank Andrea Corradini and the referees for their insightful comments on an earlier version of this paper.

References

1. Ariola, Z., Klop, J.: Equational term graph rewriting. Fundamenta Informaticae 26(3-4) (1996)
2. Barendregt, H., van Eekelen, M., Glauert, J., Kenneway, R., Plasmeijer, M.J., Sleep, M.: Term graph rewriting. In: de Bakker, J.W., Nijman, A.J., Treleaven, P.C. (eds.) PARLE 1987. LNCS, vol. 259, pp. 141–158. Springer, Heidelberg (1987)
3. Corradini, A., Gadducci, F.: A 2-categorical presentation of term graph rewriting. In: Moggi, E., Rosolini, G. (eds.) CTCS 1997. LNCS, vol. 1290, pp. 87–105. Springer, Heidelberg (1997)
4. Corradini, A., Heindel, T., Hermann, F., König, B.: Sesqui-pushout rewriting. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 30–45. Springer, Heidelberg (2006)
5. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic approaches to graph transformation - part I: Basic concepts and double pushout approach. In: Handbook of Graph Grammars, pp. 163–246 (1997)
6. Drewes, F., Hoffmann, B., Janssens, D., Minas, M., Eetvelde, N.V.: Adaptive star grammars. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 77–91. Springer, Heidelberg (2006)
7. Duval, D., Echahed, R., Prost, F.: Modeling pointer redirection as cyclic term-graph rewriting. ENTCS 176(1), 65–84 (2007)
8. Echahed, R.: Inductively sequential term-graph rewrite systems. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 84–98. Springer, Heidelberg (2008)
9. Ehrig, H., Heckel, R., Korff, M., Löwe, M., Ribeiro, L., Wagner, A., Corradini, A.: Algebraic approaches to graph transformation - part II: Single pushout approach and comparison with double pushout approach. In: Handbook of Graph Grammars, pp. 247–312 (1997)
10. Ehrig, H., Pfender, M., Schneider, H.J.: Graph-grammars: An algebraic approach. In: FOCS 1973, The University of Iowa, USA, October 15-17, pp. 167–180. IEEE, Los Alamitos (1973)
11. Hofmann, M., Jost, S.: Static prediction of heap space usage for first-order functional programs. In: POPL 2003, pp. 185–197 (2003)
12. Kennaway, R.: On "on graph rewritings". Theor. Comput. Sci. 52, 37–58 (1987)
13. Löwe, M.: Algebraic approach to single-pushout graph transformation. Theor. Comput. Sci 109(1&2), 181–224 (1993)
14. Mac Lane, S.: Categories for the Working Mathematician, 2nd edn. Springer, Heidelberg (1998)
15. Plump, D.: Term graph rewriting. In: Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G. (eds.) Handbook of Graph Grammars and Computing by Graph Transformation, vol. 2, pp. 3–61. World Scientific, Singapore (1999)
16. Power, A.J.: An abstract formulation for rewrite systems. In: Dybjer, P., Pitts, A.M., Pitt, D.H., Poigné, A., Rydeheard, D.E. (eds.) Category Theory and Computer Science. LNCS, vol. 389, pp. 300–312. Springer, Heidelberg (1989)
17. Raoult, J.C.: On graph rewriting. Theoretical Computer Science 32, 1–24 (1984)
18. Sleep, M.R., Plasmeijer, M.J., van Eekelen, M.C.J.D. (eds.): Term Graph Rewriting. Theory and Practice. J. Wiley & Sons, Chichester (1993)

An Explicit Framework for Interaction Nets

Marc de Falco*

Institut de Mathématiques de Luminy
Université de la Méditerranée
Marseille, France

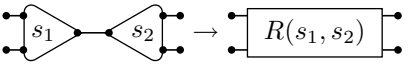
Abstract. Interaction nets are a graphical formalism inspired by Linear Logic proof-nets often used for studying higher order rewriting e.g. β-reduction. Traditional presentations of interaction nets are based on graph theory and rely on elementary properties of graph theory. We give here a more explicit presentation based on notions borrowed from Girard's *Geometry of Interaction*: interaction nets are presented as partial permutations and a composition of nets, the *gluing*, is derived from the execution formula. We then define contexts and reduction as the context closure of rules. We prove strong confluence of the reduction within our framework and show how interaction nets can be viewed as the quotient of some generalized proof-nets.

1 Introduction

Interaction nets were introduced by Yves Lafont in [Laf90] as a way to extract a model of computation from the well-behaved proof-nets of multiplicative linear logic. They have since been widely used as a formalism for the implementation of reduction strategies for the λ-calculus, providing an intuitive way to do explicit substitution [Mac98][MP98][Lip03].

Interaction nets are easy to present: a net is made of cells

with a fixed number of connection ports, depicted as big dots on the picture, one of which is distinguished and called the principal port of the cell, and of free ports, and of wires between those ports such that any port is linked by exactly one wire. Then we define reduction on nets by giving rules of the form



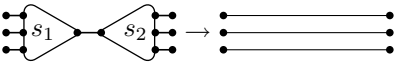
where the two cells in the left part are linked by their principal ports and the box in the right part is a net with the same free ports as the left part. Such a rule can be turned into a reduction of nets: as soon as a net contains the left part we replace it with the right part.

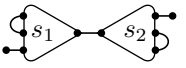
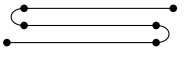
* This work is supported by the French ANR project **CHoCo.**

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 209–223, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Even though this definition is sufficient to work with interaction nets, it is too limited to reason on things like paths or observational equivalence. One of the main issues comes from the fact that we do not really know what a net is. The situation is quite similar for graphs: we cannot study them relying using drawings only without being deceived by our intuition. Thus, we are inclined to give a precise definition of a graph as a binary relation or as a set of edges.

The main issue to give such a definition for interaction nets is that it should cope with reduction. As an example consider a graph-like construction over ports and a rule



Can it be applied to the interaction net  ? If we are rigorous the left part of the rule is not exactly contained in this net as is not contained in . Perhaps we could consider this last wire as composed of three smaller ones and two temporary ports like in and the whole net after reduction would be  . But then, to get back a real interaction nets we would have to concatenate all those wires and erase the temporary ports, which would give us the net . We will refer to this process of wire concatenation as *port fusion*.

There are many works giving definitions of interaction nets giving a rigorous description of reduction. Nevertheless, they all share a common point: they deal either implicitly or externally with port fusion. In the seminal article [Laf90] a definition of nets as terms with paired variables is given, it is further refined in [FM99]. In this framework an equivalence relation on variables deals with port fusion. In [Pin00] a concrete machine is given where the computation of the equivalence relation is broken into many steps. A rigorous approach sharing some tools with ours is given in [Vau07], port fusion is done there by an external port rewriting algorithm.

Therefore, we raise the following question: can we give a definition of interaction nets allowing a simple and rigorous description of reduction encompassing port fusion, and upon which we can prove results like strong confluence? This is the aim of this paper.

Our proposition is based on the following observation. When we plug the right part of a rule in a net, new wires are defined based on a back and forth process between the original net and this right part. Such kind of interaction is key to the *geometry of interaction* [Gir89] or *game semantics* [AJM94, HO00]. The untyped nature of interaction nets makes the former a possible way to express them. To be able to do so we need to express an interaction net as some kind of partial permutation and use a composition based on the so-called *execution formula*. Such presentation of multiplicative proof-nets has been made by Jean-Yves Girard in [Gir87]. If we try to think about the fundamental actions one needs to be able to do on interaction nets, it is quite clear that we can distinguish

a *wire action* consisting in going from one port to another along a wire and the *cell action* consisting in going from one cell port to another inside the same cell. Those two actions lead to the description of a net as a pair of permutations. One might ask whether it is possible in some case to faithfully combine this pair in only one permutation, a solution to this question is what one could call a *geometry of interaction*.

The issue of port fusion is not inherent to interaction nets and can be found in other related frameworks. Diagram rewriting [Laf03] uses a well-behaved underlying category allowing mathematically the *straightening* of wires. But this is not free: the presentation is now vertically directed and lack the ease of definition of interaction nets for describing programs. Another work related to this problem is the presentation of multiplicative proof-nets by Hughes in [Hug05] where the author presents proof-nets as functions with a composition based on a categorical construction associated to traced monoidal categories [JSV96] which has been used to analyse Girard's *geometry of interaction* [AJ92, HS06]. A large part of our framework could be seen as a special case of a similar general categorical construction. Indeed, we are using the same tool as in those semantics, but our specialization to the partial injections of integers allows us to work on syntax and to stay in a completely untyped world.

2 Permutations and Partial Injections

We give here the main definitions and constructions that are going to be central to our realization of interaction nets. Those definitions are standard in the partial injections model of *geometry of interaction* [Gir87, DR95] or in the definition of the traced monoidal category PInj [HS06].

2.1 Permutations

We recall that a permutation of a set E is any bijection acting on E and we write $\mathfrak{S}(E)$ for the set of these permutations. For $\sigma \in \mathfrak{S}(E)$ we call *order* the least integer n such that $\sigma^n = id_E$, for $x \in E$ we write $\mathrm{Orb}_\sigma(x) = \{\sigma^i(x) \mid i \in \mathbb{N}\}$ and we call it the *orbit* of x, we write $\mathrm{Orbs}(\sigma)$ for the orbits of σ. If o is an orbit we write $|o|$ for its size.

We write $(c_1, \ldots, c_n)$ for the permutation sending c_i to c_{i+1}, for $i < n$, c_n to c_1 and being the identity elsewhere, we call it a *cycle* of *length* n which is also its order. Any permutation is a compound of disjoint cycles.

Let σ be a permutation of E and $\mathcal{L}$ any set, we say that σ is labelled by $\mathcal{L}$ if we have a function $l_\sigma : \mathrm{Orbs}(\sigma) \to \mathcal{L}$. We say that σ *has pointed orbits* if it is labelled by E and $\forall o \in \mathrm{Orbs}(\sigma)$ we have $l_\sigma(o) \in o$. Remark that an orbit is a sub-cycle and thus, having pointed orbits means that we have chosen a starting point in those sub-cycles.

2.2 Partial Injections

A *partial injection (of integers)* f is a bijection from a subset $\mathrm{dom}(f)$ of $\mathbb{N}$, called its *domain*, to a subset $\mathrm{codom}(f)$ of $\mathbb{N}$, called its *codomain*. We write $f : A \twoheadrightarrow B$

to say that f is any partial injection such that $\mathrm{dom}(f) = A$ and $\mathrm{codom}(f) = B$. We write $f^\star$ for the inverse of this bijection viewed as a partial injection. We call *partial permutation* a partial injection f such that $\mathrm{dom}(f) = \mathrm{codom}(f)$.

2.3 Execution

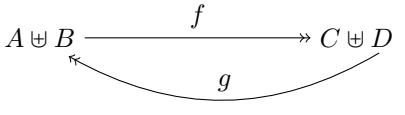
Let f be a partial injection and $E', F' \subseteq \mathbb{N}$. We write $f\restriction_{F'}^{E'}$ for the partial injection of domain $\{x \in E' \cap \mathrm{dom} f \mid f(x) \in F'\}$ ans such that $f\restriction_{F'}^{E'}(x) = f(x)$ where it is defined. We have

$$f\restriction_{F'}^{E'} : f^{-1}(F') \cap E' \twoheadrightarrow f(E') \cap F'$$

If $E = F$ and $E' = F'$ we write $f\restriction_{E'} = f\restriction_{E'}^{E'}$.

When $\mathrm{dom}(f) \cap \mathrm{dom}(g) = \emptyset$ and $\mathrm{codom}(f) \cap \mathrm{codom}(g) = \emptyset$, we say that f and g are *disjoint* and we define the sum $f + g$ and the associated refining order $\prec$ as expected. We have $\mathrm{dom}(f + g) = \mathrm{dom}(f) \uplus \mathrm{dom}(g)$ where $\uplus$ is the disjoint union.

Property 1. *Let $f : A \uplus B \twoheadrightarrow C \uplus D$ and $g : D \twoheadrightarrow B$ a situation depicted by the following diagram*



i For all $n \in \mathbb{N}$, the partial injection from A to C

$$\mathsf{Ex}_n(f,g) = f\restriction_C^A + (fgf)\restriction_C^A + \cdots + (f(gf)^n)\restriction_C^A$$

is well defined.

ii $(\mathsf{Ex}_n(f,g))_{n\in\mathbb{N}}$ is an increasing sequence of partial injections with respect to $\prec$, whose limit, the increasing union, is noted $\mathsf{Ex}(f,g)$.

iii If $\mathrm{dom}(f)$ is finite the sequence $(\mathsf{Ex}_n(f,g))_n$ is stationary and

$$\mathsf{Ex}(f,g) : A \twoheadrightarrow C$$

Fig. 1 gives a graphical presentation of execution.

Proof. i) To assert the validity of the sum all we have to have show is that $\forall i \neq j \in \mathbb{N}$:

$$(f(gf)^i)(A) \cap (f(gf)^j)(A) \cap C = \emptyset$$
$$(f(gf)^i)^{-1}(C) \cap (f(gf)^j)^{-1}(C) \cap A = \emptyset$$

Suppose there is a $x \in (f(gf)^i)(A) \cap (f(gf)^j)(A) \cap C$, we set y and $z \in A$ such that $x = f(gf)^i(y) = f(gf)^j(z)$. We can further suppose that $i < j$, and we have $y = (gf)^{j-i}(z) \in B$, which is contradictory as $y \in A$ and $A \cap B = \emptyset$.

The other equality is proved in the same way.

ii) Let $n \leq m \in \mathbb{N}$ and $x \in \mathrm{dom}(\mathsf{Ex}_n(f,g))$, by definition of the sum there exists a unique k such that $\mathsf{Ex}_n(f,g)(x) = (f(gf)^k)(x)$. But then $x \in$

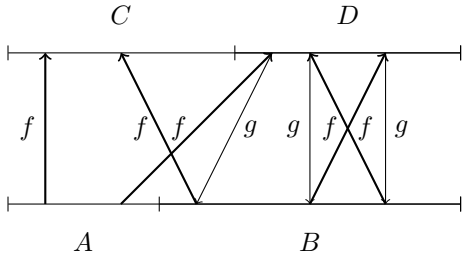


Fig. 1. Representation of $\mathsf{Ex}(f,g)$ with the notations of property 1

$\mathrm{dom}(\mathsf{Ex}_m(f,g))$ and the uniqueness of k asserts that $\mathsf{Ex}_m(f,g)(x) = (f(gf)^k)(x)$. Thus, $\mathsf{Ex}_m(f,g)$ is a refinement of $\mathsf{Ex}_n(f,g)$.

iii) Suppose there is a $x \in A - \mathrm{dom}(\mathsf{Ex}(f,g))$, then we should have for all k, $(f(gf)^k)(x) \in D$ or else $\mathsf{Ex}(f,g)(x)$ would be defined. But D being finite, there exists $n \leq m$ such that $(f(gf)^n)(x) = (f(gf)^m)(x)$ and we get $x = (gf)^{m-n}(x) \in B$ which is contradictory. A simple argument on cardinal show then that $\mathrm{codom}(\mathsf{Ex}(f,g)) = C$. □

Theorem 2 (Associativity of execution).

Let $A \uplus B \uplus C \overset{f}{\underset{g}{\rightleftarrows}} D \uplus E \uplus F$ (with h : $F \to A \uplus B \uplus C$) *be three partial injections.*

We have $\forall n \in \mathbb{N}$

$$\mathsf{Ex}_n(\mathsf{Ex}_n(f,g),h) = \mathsf{Ex}_n(f,g+h) = \mathsf{Ex}_n(\mathsf{Ex}_n(f,h),g)$$

and thus

$$\mathsf{Ex}(\mathsf{Ex}(f,g),h) = \mathsf{Ex}(f,g+h) = \mathsf{Ex}(\mathsf{Ex}(f,h),g)$$

Proof. Let $p \in \mathrm{dom}(\mathsf{Ex}_n(f,g+h))$, there exists $m \leq n \in \mathbb{N}$ such that

$$\begin{aligned}
\mathsf{Ex}_n(f,g+h)(p) &= f((g+h)f)^m(p) \\
&= (f(gf)^{i_1})h \dots h(f(gf)^{i_k})(p) \text{ with } i_1 + \dots + i_k + k - 1 = m \\
&= (\mathsf{Ex}_n(f,g)h\mathsf{Ex}_n(f,g) \dots h\mathsf{Ex}_n(f,g))(p) \\
&= (\mathsf{Ex}_n(f,g)(h\mathsf{Ex}_n(f,g))^{k-1})(p) \\
&= \mathsf{Ex}_n(\mathsf{Ex}_n(f,g),h)(p)
\end{aligned}$$

By commutativity of $+$ we get the other equality. These equalities are directly transmitted to Ex. □

This theorem is of utter significance, it is a completely localized version of Church-Rosser property. Indeed, we will see later that confluence results are corollary of this theorem.

2.4 w-Permutations and Ex-Composition

We call *w-permutation* an involutive partial permutation of finite domain.

Let σ and τ be disjoint w-permutations and let f be a partial injection with $\mathrm{dom}(f) \subseteq \mathrm{dom}(\sigma)$ and $\mathrm{codom}(f) \subseteq \mathrm{dom}(\tau)$. We call the Ex_0-*composition* of σ and τ along f the partial permutation

$$\sigma \overset{f}{\leftrightsquigarrow}_0 \tau = \mathsf{Ex}(\sigma + \tau, f + f^\star)$$

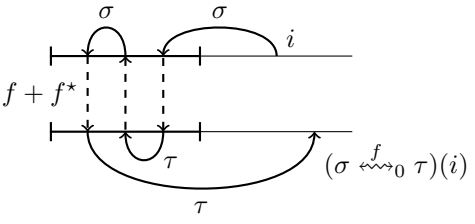


Fig. 2. Representation of the Ex_0-composition $\sigma \overset{f}{\leftrightsquigarrow}_0 \tau$

Fig. 2 gives a representation of this composition.

Property 3. *$\sigma \overset{f}{\leftrightsquigarrow}_0 \tau$ is a w-permutation.*

Proof. Let x be an element of $\mathrm{dom}(\sigma + \tau)$, there exists n such that $(\sigma \overset{f}{\leftrightsquigarrow}_0 \tau)(x) = (f + f^\star)[(\sigma + \tau)(f + f^\star)]^n(x)$. Note that $(f + f^\star)^\star = f + f^\star$ and $(\sigma+\tau)^\star = \sigma+\tau$, and thus, we have $((f+f^\star)[(\sigma+\tau)(f+f^\star)]^n)^\star = [(f+f^\star)(\sigma+\tau)]^n(f + f^\star) = (f + f^\star)[(\sigma + \tau)(f + f^\star)]^n$. So $(\sigma \overset{f}{\leftrightsquigarrow}_0 \tau)^2(x) = x$. □

To define the final Ex-composition we want to recover the fix-points hidden by Ex in order to get the usual notion of *loops*. Suppose that there is an x_0 such that:

$$x_0 \xrightarrow{\sigma+\tau} y_0 \xrightarrow{f+f^\star} x_1 \ldots \xrightarrow{\sigma+\tau} y_n \xrightarrow{f+f^\star} x_n = x_0$$

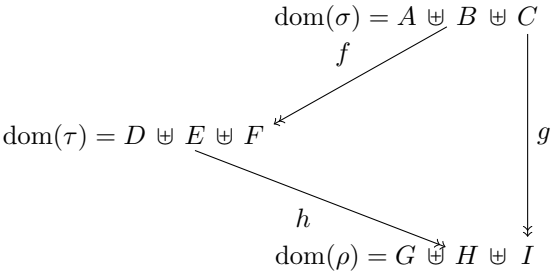
Everything being involutive, this loop is reversible and we get a new loop

$$y_0 \xleftarrow{f+f^\star} x_1 \ldots \xleftarrow{\sigma+\tau} y_n \xleftarrow{f+f^\star} x_0 \xleftarrow{\sigma+\tau} y_0$$

We say that the set $\{x_0, y_0, \ldots, x_{n-1}, y_n\}$ forms a *double orbit*, and we it can be fully reconstructed from any of its element. Therefore, recall that all this points are integers, and let R be a set comprised of the least element of each double orbit. We can define the Ex-composition, written $\sigma \overset{f}{\leftrightsquigarrow} \tau$, extending $\sigma \overset{f}{\leftrightsquigarrow}_0 \tau$ on R by $(\sigma \overset{f}{\leftrightsquigarrow} \tau)(r) = r$ for $r \in R$.

We can now give a direct corollary of theorem 2, stating some kind of associativity for the Ex-composition.

Corollary 4. *Let σ, τ, ρ be pairwise disjoint w-permutations with*



We have $\sigma \overset{f+g}{\leftrightsquigarrow} (\tau \overset{h}{\leftrightsquigarrow} \rho) = (\sigma \overset{f}{\leftrightsquigarrow} \tau) \overset{g+h}{\leftrightsquigarrow} \rho = (\sigma \overset{g}{\leftrightsquigarrow} \rho) \overset{f+h}{\leftrightsquigarrow} \tau$. *When* $h = 0$ *we get* $\sigma \overset{f+g}{\leftrightsquigarrow} (\tau + \rho) = (\sigma \overset{f}{\leftrightsquigarrow} \tau) \overset{g}{\leftrightsquigarrow} \rho = (\sigma \overset{g}{\leftrightsquigarrow} \rho) \overset{f}{\leftrightsquigarrow} \tau$.

3 The Statics of Interaction Nets

We fix a countable set $\mathcal{S}$, whose elements are called *symbols*, and a function $\alpha : \mathcal{S} \to \mathbb{N}$, the *arity*. We will define nets atop $\mathbb{N}$ and in this context an integer will be called a *port*.

Definition 5. *An interaction net is an ordered pair $R = (\sigma_w, \sigma_c)$ where:*

- *σ_w is a w-permutation. We write $P_l(R)$ for the fixed points of σ_w and $P(R)$ for the others.*
- *σ_c is a partial permutation of $P(R)$ with pointed orbits and labelled by $\mathcal{S}$ in such a way that $\forall o \in Orbs(\sigma_c), |o| = \alpha(l(o))$ where l is the labelling function.*

The elements of $P_l(R)$ are called *loops* and the other orbits of σ_w, which are necessarily of length 2, are called *wires*. The domain of σ_w is called the *carrier* of the net. We write $P_c(R) = \mathrm{dom}(\sigma_c)$, whose elements are called *cell ports*, and $P_f(R) = P(R) - P_c(R)$, whose elements are called *free ports*.

An orbit of σ_c is called a *cell*. We write pal for the pointing function of σ_w. Let c be a cell, $\mathrm{pal}(c)$ is its *principal port* and for $i < |c|$ the element $(\sigma_c^i \circ \mathrm{pal})(c)$ is its *ith auxiliary port*.

Note that a port is present in exactly one wire and at most one cell.

3.1 Representation

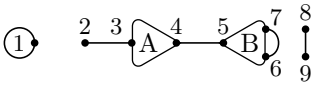
Nets admit a very natural representation. We shall draw a cell of symbol A as a triangle

where the principal port is the dot on the apex and auxiliary ports are lined up on the opposing edge. We draw free ports as points. To finish the drawing we add a line between any two ports connected by a wire, and draw circles for loops.

As an example consider the net $R = (\sigma_w, \sigma_c)$ with

$$\sigma_w = (1)(2\ 3)(4\ 5)(6\ 7)(8\ 9) \text{ and } \sigma_c = (\overset{\bullet}{4}\ \ 3)_A(\overset{\bullet}{5}\ \ 6\ 7)_B$$

where permutations are given by cycle decomposition and $(\overset{\bullet}{c_1}\ c_2\ \dots\ c_n)_S$ is a cell of point c_1 and symbol S. This net will have the representation



3.2 Morphisms of Nets and Renaming

Definition 6. *Let $R = (\sigma_w, \sigma_c)$ and $R' = (\sigma'_w, \sigma'_c)$ be two interaction nets. The function $f : P(R) \mapsto P(R')$ is a* morphism from R to R' *iff*

$$f \circ \sigma_w = \sigma'_w \circ f, \quad f(P_c(R)) \subseteq P_c(f(R')),$$

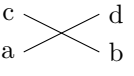
$$\forall p \in P_c(R), (f \circ \sigma_c)(p) = (\sigma'_c \circ f)(p),$$

and $\forall o \in Orbs(\sigma_c)$ we have $(f \circ pal)(o) = (pal \circ f)(o)$ and $l(o) = (l \circ f)(o)$. When f is the identity on $P_f(R)$ it is said to be an internal morphism.

Let us detail a bit more this definition. We note that for any two partial permutations σ and τ, the equation $f \circ \sigma = \tau \circ f$ induces that a $o \in \mathrm{Orbs}(\sigma)$ is mapped to an element $f(o) \in \mathrm{Orbs}(\tau)$ such that $|f(o)|$ is a divisor of $|o|$.

In this case a loop is sent to a loop, a wire to a loop or a wire, and a cell to another cell. The last two equations say that the principal port of cell is mapped to a principal port, and symbols are preserved. So a cell is mapped to a cell of same arity, and each port is mapped to the same type of port. Moreover only a wire linking free ports can be mapped to a loop or any kind of wire. As soon as the wire is linking one cell port the third condition on the morphism must send it to a wire of the same type.

With this remark, it is natural to call *renaming* (resp. *internal renaming*) an isomorphism (resp. internal isomorphism). An isomorphism class captures interaction nets as they are drawn on paper. On the other hand, an internal isomorphism class corresponds to interaction nets drawn where we have also given distinct names to free ports, hence the name internal. This is an important notion because the drawing is the same as Whereas the drawing

is different from 

.

Remark 7. *Given the fact that nets have finite carriers we can always consider that two nets have disjoint carriers up to renaming.*

4 Tools of the Trade

4.1 Gluing and Cutting

Definition 8. *Let $R = (\sigma_w, \sigma_c)$ and $R' = (\sigma'_w, \sigma'_c)$ be two nets with disjoint carriers* [1] *and let $f : P_f(R) \hookrightarrow P_f(R')$. We call* gluing of R and R' along f *the net $R \overset{f}{\leftrightsquigarrow} R' = (\sigma_w \overset{f}{\leftrightsquigarrow} \sigma'_w, \sigma_c + \sigma'_c)$.*

[1] Which is not a loss of generality thanks to remark 7.

From this definitions we get the following obvious facts:

$$P(R \overset{f}{\leftrightsquigarrow} R') = (P(R) - \mathrm{dom}(f)) \uplus (P(R') - \mathrm{codom}(f))$$
$$P_c(R \overset{f}{\leftrightsquigarrow} R') = P_c(R) \uplus P_c(R')$$
$$P_f(R \overset{f}{\leftrightsquigarrow} R') = (P_f(R) - \mathrm{dom}(f)) \uplus (P_f(R') - \mathrm{codom}(f))$$
$$R \overset{f}{\leftrightsquigarrow} R' = R' \overset{f^\star}{\leftrightsquigarrow} R$$

For the special case of gluing where $f = 0$ we have $R \overset{0}{\leftrightsquigarrow} R' = (\sigma_w + \sigma'_w, \sigma_c + \sigma'_c)$, we write this special kind of gluing $R + R'$, it is the so-called parallel composition of the two nets. Fig. 3 gives a representation of gluing.

Property 9. *If $R = R \overset{f}{\leftrightsquigarrow} R'$ then $f = 0$ and $R' = \mathbf{0} = (0, 0)$. If $\mathbf{0} = R \overset{f}{\leftrightsquigarrow} R'$ then $f = 0$ and $R = R' = \mathbf{0}$.*

Proof. We will only prove the first assertion, the second being similar. It is a direct consequence of the previous facts, R' must have no cells, no free ports and no loops. The only net having this property is the empty net $\mathbf{0}$. □

We can get some kind of associativity property for gluing.

Property 10. *Let $R = (\sigma_w, \sigma_c)$, $S = (\tau_w, \tau_c)$ and $T = (\rho_w, \rho_c)$ be nets of disjoint carriers and let f, g and h be partial injections satisfying the diagram of corollary 4 with respect to σ_w, τ_w and ρ_w.*

We have $R \overset{f+g}{\leftrightsquigarrow} (S \overset{h}{\leftrightsquigarrow} T) = (R \overset{f}{\leftrightsquigarrow} S) \overset{g+h}{\leftrightsquigarrow} T = (R \overset{g}{\leftrightsquigarrow} T) \overset{f+h}{\leftrightsquigarrow} S$.

Proof. The wire part of the equality is a restriction of corollary 4 and the cell part is the associativity of $+$. □

The following corollary will often be sufficient.

Corollary 11. *If we have a decomposition $R_0 = R \overset{f}{\leftrightsquigarrow} (S \overset{g}{\leftrightsquigarrow} T)$ then there exists f_S, f_T such that $R_0 = (R \overset{f_S}{\leftrightsquigarrow} S) \overset{g+f_T}{\leftrightsquigarrow} T$.*

We can use the gluing to define dually the notion of cutting a subnet of an interaction net.

Definition 12. *Let R be a net, we call* cutting of R *a triple (R_1, f, R_2) such that $R = R_1 \overset{f}{\leftrightsquigarrow} R_2$. Any net R' appearing in a cutting of R is called a* subnet of R, *noted $R' \subseteq R$.*

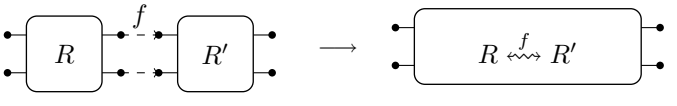


Fig. 3. Representation of the gluing of two interaction nets

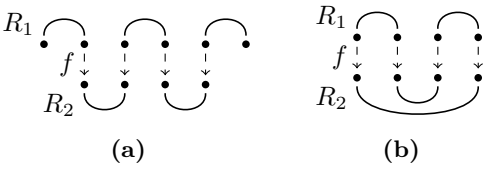


Fig. 4. Representation of two special cuttings: (a) a cutting of a single wire and (b) a cutting of a loop

The Fig. 4 gives an example of cutting. The fact that we can cut many times a wire or that we can divide a loop in many wires hints at the complexity behind these definitions.

Property 13. *The relation $\subseteq$ is an ordering of nets.*

Proof. The relation $\subseteq$ is **reflexive**: $R = R \overset{0}{\rightsquigarrow} \mathbf{0}$ and thus, $R \subseteq R$.

It is **antisymmetric**: let R_1 and R_2 be nets such that $R_1 \subseteq R_2$ and $R_2 \subseteq R_1$. We have $R_1 = R_2 \overset{f}{\rightsquigarrow} R_2'$ and $R_2 = R_1 \overset{g}{\rightsquigarrow} R_1'$.

So $R_2 = (R_2 \overset{f}{\rightsquigarrow} R_2') \overset{g}{\rightsquigarrow} R_1'$. By applying the corollary 11 we get $R_2 = R_2 \overset{f_1}{\rightsquigarrow} (R_2' \overset{g+f_2}{\rightsquigarrow} R_1')$. and by applying the property 9 twice we get $R_2' = R_1' = \mathbf{0}$. So $R_1 = R_2$.

And it is **transitive**: let $R \subseteq S \subseteq T$, then $S = R \overset{f}{\rightsquigarrow} R'$ and $T = S \overset{g}{\rightsquigarrow} S'$, so $T = (R \overset{f}{\rightsquigarrow} R') \overset{g}{\rightsquigarrow} S'$. By applying the corollary 11 we have $T = R \overset{f_1}{\rightsquigarrow} (R' \overset{g+f_2}{\rightsquigarrow} S')$, that is to say $R \subseteq T$. □

4.2 Interfaces and Contexts

To define reduction by using the subnet relation, it would be easier if we could refer implicitly to the identification function in a gluing. As an intuition, consider terms contexts with multiple holes, to substitute completely such contexts we could give a function from holes to terms and fill them accordingly. But a more natural definition would be to give a distinct number to each hole and to fill based on a list of terms. The substitution would give the first term to the first hole, and so on. The following definition is a direct transposition of this idea in the framework of interaction nets.

Definition 14. *We call* interface of a net R *a subset $I = \{p_1, \ldots, p_n\}$ of $P_f(R)$ together with a linear ordering, the length of the order chain $p_1 < \cdots < p_n$ is called the* size. *We say that R contains the interface I, noted $I \subset R$. An interface is* canonical *if it contains all the free ports of a net.*

Let I and I' be disjoint interfaces of the same net, we write II' the union of these subsets ordered by the concatenation of the two order chains. Precisely $x \leq_{II'} y \iff x \leq_I y$ or $x \leq_{I'} y$ or $x \in I \wedge y \in I'$.

Let I and I' be two interfaces of same order, there exists one and only order-preserving bijection from I to I' that we write $\rho(I, I')$ and call the chord between I and I'.

We call context *a couple* (R, I) *where* I *is an interface contained in the net* R*, it is written* R^I*.*

Let R^I *and* $R'^{I'}$ *be two contexts with interfaces of same order, we write*

$$R^I \leftrightsquigarrow R'^{I'} = R \overset{\rho(I,I')}{\leftrightsquigarrow} R'$$

In the following when we write $R^I \leftrightsquigarrow R'^{I'}$ we implicitly assume that I and I' are of same size.

We now can state commutativity of gluing directly, the proof being trivial.

Property 15 (Commutativity of gluing). $R^I \leftrightsquigarrow R'^{I'} = R'^{I'} \leftrightsquigarrow R^I$

The following trivial fact asserts that any gluing can be seen as a context gluing.

Fact 16. *Let* $R \overset{f}{\leftrightsquigarrow} R'$ *be a gluing, there exist interfaces* $I \subset R$ *and* $I' \subset R'$ *of same order such that* $R \overset{f}{\leftrightsquigarrow} R' = R^I \leftrightsquigarrow R'^{I'}$*.*

Corollary 17. $R_1 \subseteq R \iff \exists I_1, R_2, I_2$ *such that* $R = {R_1}^{I_1} \leftrightsquigarrow {R_2}^{I_2}$

We can now restate the corollary 11 with interfaces:

Corollary 18. *For all nets* R, S, T *and interfaces* I, J, K, L*, there exists interfaces* I', J', K', L' *such that*

$$R^I \leftrightsquigarrow (S^J \leftrightsquigarrow T^K)^L = (R^{I'} \leftrightsquigarrow S^{J'})^{L'} \leftrightsquigarrow T^{K'}$$

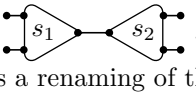
5 Dynamics

Definition 19. *Let* s_1 *and* s_2 *be symbols. We call* interaction rule for (s_1, s_2) *a couple* $({R_r}^{I_r}, {R_p}^{I_p})$ *where*

$$R_r = \begin{pmatrix} (b\ c)(a_1\ b_1)\dots(a_n\ b_n)(c_1\ d_1)\dots(c_m\ d_m), \\ (\overset{\bullet}{b}\ b_1\ \dots\ b_n)_{s_1}(\overset{\bullet}{c}\ c_1\ \dots\ c_m)_{s_2} \end{pmatrix}$$

and I_r *and* I_p *are both canonical – comprised of all free ports – and of same size.*

Let $\mathcal{R} = ({R_r}^{I_r}, {R_p}^{I_p})$ *be a rule we call* reduction by $\mathcal{R}$ *the binary relation* $\xrightarrow{\mathcal{R}}$ *on nets such that for all renaming* α *and* β*, and for all net* S *with* $S = R^I \leftrightsquigarrow \alpha(R_r)^{\alpha(I_r)}$ *we set* $S \xrightarrow{\mathcal{R}} S'$ *where* $S' = R^I \leftrightsquigarrow \beta(R_p)^{\beta(I_p)}$*.*

The net R_r has the representation 

. Remark that the reduction is defined as soon as a net contains a renaming of the redex R_r. This reduction appears to be non-deterministic but it is only the expansion of a deterministic reduction to cope with all possible renamings.

Property 20. *Let R be a net and $\mathcal{R}_1$, $\mathcal{R}_2$ be two interaction rules applicable on R on distinct redexes such that $R_1 \xleftarrow{\mathcal{R}_1} R \xrightarrow{\mathcal{R}_2} R_2$ and all the ports both in R_1 and R_2 are also in R. There exists a net R' such that $R_1 \xrightarrow{\mathcal{R}_2} R' \xleftarrow{\mathcal{R}_1} R_2$.*

Proof. For $i = 1, 2$, set $\mathcal{R}_i = ({R_{r,i}}^{I_{r,i}}, {R_{p,i}}^{I_{p,i}})$. The shape of redexes allow us to assert that if they are distinct then they are disjoint. As R contains both a redex $\alpha_1(R_{r,1})$ and a redex $\alpha_2(R_{r,2})$, then we can deduce that $\alpha_1(R_{r,1}) + \alpha_2(R_{r,2}) \subseteq R$. More precisely we have

$$R = (\alpha_1(R_{r,1}) + \alpha_2(R_{r,2}))^{\alpha_1(I_{r,1})\alpha_2(I_{r,2})} \leftrightsquigarrow {R_0}^I$$

We get

$$R_1 = (\beta_1(R_{p,1}) + \alpha_2(R_{r,2}))^{\beta_1(I_{p,1})\alpha_2(I_{r,2})} \leftrightsquigarrow {R_0}^I$$

for a renaming β_1, and the same kind of expression for R_2. It is straightforward to check that the net

$$R' = (\beta_1(R_{p,1}) + \beta_2(R_{p,2}))^{\beta_1(I_{p,1})\beta_2(I_{p,2})} \leftrightsquigarrow {R_0}^I$$

satisfies the conclusion by applying property 10. The very existence of this net relies on the disjointness of the $\beta_i(R_{p,i})$ which is ensured by the hypothesis on ports contained in both R_1 and R_2. □

Corollary 21. *Let $\mathcal{L}$ be a set of rules such that for any pair of symbols there is at most one rule over them. The reduction $\xrightarrow{\mathcal{L}} = \bigcup_{\mathcal{R}\in\mathcal{L}} \xrightarrow{\mathcal{R}}$ is strongly confluent up to a renaming.*

By *up to a renaming* we mean that we might have to rename one of the nets in a critical pair before joining them. This is due to the disjointness condition in property 20. Remark that we can always substitute one of the branch of the critical pair by another instance of the same rule on the same redex in such a way that this condition is ensured.

6 Interaction Nets are the Ex-Collapse of Axiom/Cut Nets

We introduce now a notion of nets lying between proof-nets of multiplicative linear logic and interaction nets. When we plug directly two interaction nets a complex process of wire simplification occurs. When we plug two proof-nets we only add special wires called *cuts* and we have an external notion of reduction performing such simplification. In this section we define nets with two kinds of wires: *axioms* and *cuts*. Those nets allow us to give a precise account of the folklore assertion that interaction nets are a quotient of multiplicative proof-nets.

6.1 Definition and Juxtaposition

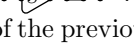
Definition 22. *An* Axiom/Cut net, AC net *for short, is a tuple $R = (\sigma_A, \sigma_C, \sigma_c)$ where:*

- σ_A *and* σ_C *are w-permutations of finite domain included in* $\mathbb{P}$*, such that* $\mathrm{dom}(\sigma_C) \subseteq \mathrm{dom}(\sigma_A)$*,* σ_C *has no fixed points and if* $(a\ b)$ *is an orbit of* σ_C *then there exists* $c \neq a$ *and* $d \neq b$ *such that* $(c\ a)$ *and* $(b\ d)$ *are orbits of* σ_A *We write* $P_l(R)$ *for the fixed points of* σ_A *and* $P(R) = \mathrm{dom}(\sigma_A) - \mathrm{dom}(\sigma_C) - P_l(R)$*.*
- σ_c *is an element of* $\mathfrak{S}(P_c(R))$*, where* $P_c(R) \subseteq P(R)$*, has pointed orbits and is labelled by* $\mathcal{S}$ *in such a way that* $\forall o \in Orbs(\sigma_c), |o| = \alpha(l(o))$ *where* l *is the labelling function.*

The orbits of σ_C, called *cuts*, are some kind of undirected unary cells linking orbits of σ_A, called *axioms*.

We directly adapt the representation of interaction nets to AC nets by displaying σ_c as double edges. For example the AC net $R = (\sigma_A, \sigma_C, \sigma_c)$ with

$$\sigma_A = (1\ 2)(3\ 4)(5\ 6), \sigma_C = (2\ 3), \sigma_c = (\overset{\bullet}{4}\ \ 5)_s$$

will be represented by  .

We can adapt most of the previous definitions for those nets, most importantly free ports, interfaces and contexts. The nice thing about AC nets is that they yield a very simple composition.

Definition 23. *Let* $R^I = (\sigma_A, \sigma_C, \sigma_c)$ *and* $R'^{I'} = (\tau_A, \tau_C, \tau_c)$ *be two contexts on AC nets with disjoint carriers, with* $I = i_1 > \cdots > i_n$ *and* $I' = i'_1 > \cdots > i'_n$*.*

We call juxtaposition of R^I and $R'^{I'}$ *the AC net*

$$R^I \leftrightarrow R'^{I'} = (\sigma_A + \tau_A, \sigma_C + \tau_C + (i_1\ i'_1) \ldots (i_n\ i'_n), \sigma_c + \tau_c)$$

The juxtaposition is from the logical point of view a generalized cut, and its interpretation in terms of permutation is exactly the definition made by Jean-Yves Girard in [Gir87].

6.2 Ex-collapse

Property 24. *Let* $R = (\sigma_A, \sigma_C, \sigma_c)$ *be an AC net and* $f : \mathbb{P} \hookrightarrow \mathbb{P}$ *be such that* $\mathrm{dom}(\sigma_C) = \mathrm{dom}(f)$ *and* $\mathrm{codom}(f) \cap \mathrm{dom}(\sigma_A) = \emptyset$*.*

The couple $(\sigma_A \overset{f}{\leftrightsquigarrow} f \circ \sigma_C \circ f^\star, \sigma_c)$*, is an interaction net.*

It does not depend on f *and we call it the* Ex*-collapse of* R*, noted* $\mathsf{Ex}(R)$*.*

For the definition of the Ex-composition to be correct, we have to delocalize σ_C to a domain disjoint from $\mathrm{dom}(\sigma_A)$. The Ex-collapse amounts to replace any maximal chain $a_1 \xrightarrow{\sigma_A} b_1 \xrightarrow{\sigma_C} a_2 \ldots b_{n-1} \xrightarrow{\sigma_A} a_n$ by a chain $a_1 \xrightarrow{\sigma_A} b_1 \xrightarrow{f} f(b_1) \xrightarrow{f\circ\sigma_C\circ f^\star} f(a_2) \xrightarrow{f^\star} \ldots b_{n-1} \xrightarrow{\sigma_A} a_n$ and then to compute the Ex-composition to get $a_1 \xrightarrow{\sigma_A \overset{f}{\leftrightsquigarrow} \sigma_C} a_n$.

Proof. It comes from the definitions of the Ex-composition and from property 3. □

Property 25. *For each interaction net R there exists a unique AC net R' of the form $(\sigma_A, 0, \sigma_c)$ such that $\mathsf{Ex}(R') = R$. R' is said to be* cutfree.

Proof. If $R = (\tau_w, \tau_c)$ we only have to take $R' = (\tau_w, 0, \tau_c)$. Uniqueness comes from the fact that $\sigma \overset{0}{\leftrightsquigarrow} 0 = \sigma$. □

Definition 26. *Let R and R' be two AC nets, we say that* R and R' are Ex-equivalent*, noted $R \overset{\leftrightsquigarrow}{\sim} R'$ when $\mathsf{Ex}(R) = \mathsf{Ex}(R')$.*

We have an obvious correspondence between juxtaposition and gluing.

Property 27. $\mathsf{Ex}(R^I \leftrightarrow R'^{I'}) = \mathsf{Ex}(R)^I \leftrightsquigarrow \mathsf{Ex}(R')^{I'}$

Therefore we can claim that

Interaction nets are the quotient of AC nets by $\overset{\leftrightsquigarrow}{\sim}$.

7 Conclusion

We could not include all of the possible extensions of our framework in this paper. Most of this results can be found in [dF09]. We have: 1) a double-pushout approach to reduction by mean of the category of interaction nets and morphisms (2) a rigorous definition of boxes as a partial labelling of cells (3) definitions of paths in a net, path reduction and proofs that the path reduction is strongly confluent (4) a full implementation in Haskell.

References

[AJ92] Abramsky, S., Jagadeesan, J.: New foundations for the geometry of interaction. In: Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science, 1992. LICS 1992, pp. 211–222 (1992)

[AJM94] Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF (extended abstract). In: Hagiya, M., Mitchell, J.C. (eds.) TACS 1994. LNCS, vol. 789, pp. 1–15. Springer, Heidelberg (1994)

[dF09] de Falco, M.: Géométrie de l'Interaction et Réseaux d'Interaction Différentiels. Thèse de doctorat, Université d'Aix-Marseille 2 (2009)

[DR95] Danos, V., Regnier, L.: Proof-nets and the Hilbert space. In: Girard, J.-Y., Lafont, Y., Regnier, L. (eds.) Advances in Linear Logic. London Mathematical Society Lecture Note Series, vol. 222. Cambridge University Press, Cambridge (1995)

[FM99] Fernandez, M., Mackie, I.: A calculus for interaction nets. In: Nadathur, G. (ed.) PPDP 1999. LNCS, vol. 1702, pp. 170–187. Springer, Heidelberg (1999)

[Gir87] Girard, J.-Y.: Multiplicatives. In: Lolli (ed.) Logic and Computer Science: New Trends and Applications, Torino, pp. 11–34. Università di Torino. Rendiconti del seminario matematico dell'università e politecnico di Torino (1987) (special issue)

[Gir89] Girard, J.-Y.: Geometry of interaction I: an interpretation of system F. In: Valentini, F.B., Zanardo (eds.) Proceedings of the Logic Colloquium 88, Padova, vol. 88, pp. 221–260. North-Holland, Amsterdam (1989)

[HO00] Hyland, M., Ong, L.: On full abstraction for PCF. Information and Computation 163, 285–408 (2000)

[HS06] Haghverdi, E., Scott, P.: A categorical model for the geometry of interaction. Theoretical Computer Science 350(2-3), 252–274 (2006)

[Hug05] Hughes, D.: Simple multiplicative proof nets with units. Archived as math.LO/0507003 at arXiv.org (submitted) (March 2005)

[JSV96] Joyal, A., Street, R., Verity, D.: Traced Monoidal Categories. In: Proc. Comb. Phil. Soc., vol. 119, pp. 447–468 (1996)

[Laf90] Lafont, Y.: Interaction nets. In: Proceedings of the 17^{th} Annual ACM Symposium on Principles of Programming Languages, pp. 95–108. ACM Press, San Francisco (1990)

[Laf03] Lafont, Y.: Towards an Algebraic Theory of Boolean Circuits. Journal of Pure and Applied Algebra 184(2-3), 257–310 (2003)

[Lip03] Lippi, S.: Encoding left reduction in the λ-calculus with interaction nets. Mathematical Structures in Computer Science 12(06), 797–822 (2003)

[Mac98] Mackie, I.: YALE: yet another lambda evaluator based on interaction nets. In: Proceedings of the third ACM SIGPLAN international conference on Functional programming, pp. 117–128. ACM, New York (1998)

[MP98] Mackie, I., Pinto, J.S.: Compiling the Lambda Calculus into Interaction Combinators. In: Logical Abstract Machines workshop (1998)

[Pin00] Pinto, J.S.: Sequential and concurrent abstract machines for interaction nets. In: Tiuryn, J. (ed.) FOSSACS 2000. LNCS, vol. 1784, pp. 267–282. Springer, Heidelberg (2000)

[Vau07] Vaux, L.: Lambda-calcul différentiel et logique classique. Thèse de doctorat, Université de la Méditerranée (2007)

Dual Calculus with Inductive and Coinductive Types

Daisuke Kimura[1] and Makoto Tatsuta[2]

[1] University of Tokyo, 7-3-1 Hongo, Tokyo 113-8656, Japan
kimura@lyon.is.s.u-tokyo.ac.jp
[2] National Institute of Informatics, 2-1-2 Hitotsubashi, Tokyo 101-8430, Japan
tatsuta@nii.ac.jp

Abstract. This paper gives an extension of Dual Calculus by introducing inductive types and coinductive types. The same duality as Dual Calculus is shown to hold in the new system, that is, this paper presents its involution for the new system and proves that it preserves both typing and reduction. The duality between inductive types and coinductive types is shown by the existence of the involution that maps an inductive type and a coinductive type to each other. The strong normalization in this system is also proved. First, strong normalization in second-order Dual Calculus is shown by translating it into second-order symmetric lambda calculus. Next, strong normalization in Dual Calculus with inductive and coinductive types is proved by translating it into second-order Dual Calculus.

1 Introduction

Dual Calculus DC is a type system which represents computation induced by cut elimination in the classical sequent calculus LK by using terms and their reduction [16,17]. DC has two nice properties: computation in classical logic, and duality.

DC can formalize computation in classical logic such as catch/throw and continuation in the same way as other systems based on classical logic such as $\lambda\mu$-calculus [11] and $\overline{\lambda}\mu\tilde{\mu}$ [3].

DC faithfully inherits the duality of the classical sequent calculus LK. DC is based on only negation, conjunction, and disjunction. In LK, conjunction and disjunction are mapped to each other by the involution which maps A to $\neg A$. This property holds also in DC. By this transformation, the sequent $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ is mapped to the sequent $\neg B_1, \ldots, \neg B_m \vdash \neg A_1, \ldots, \neg A_n$ in LK. In DC, an antecedent gives typing of coterms which mean computation of continuation, and a succedent gives typing of terms which give ordinary computation. In DC, this transformation is extended to terms and coterms and they are mapped to each other. Other computational systems do not have this duality since they are based on implication.

DC enables us to show the duality in a clear syntactic way. The duality in computation in classical logic has been studied in category theory by investigating control category [14]. The duality between ordinary computation and computation of continuation, the duality between call-by-value computation and call-by-name computation [14], and the duality between the fixed point operator and the loop operator [5] have been shown. DC enables us to prove those results in a type system in a clear syntactic way. Moreover DC enables us also to study (1) reduction, (2) non-extensional systems without η-rules, and (3) direct extension to programming languages. Wadler [16,17] showed the

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 224–238, 2009.
© Springer-Verlag Berlin Heidelberg 2009

duality between ordinary computation and computation of continuation, and the duality between call-by-value computation and call-by-name computation in a syntactical way by using Dual Calculus. Kimura [6] showed the duality between the fixed point operator and the loop operator in a type system by using Dual Calculus. The first author of this paper showed the duality of reduction between call-by-value computation and call-by-name computation in $\lambda\mu$-calculus by using DC [7] to answer the open question presented in the invited talk at RTA2005 [17].

Inductive definitions are important in both mathematical logic and computer science. Inductive definitions strengthen expressiveness of logical systems [2]. They are indispensable in programming and program verification [10,13] for handling recursive data structures such as lists and trees, and specification of recursive programs. Coinductive definitions are also important since they can represent streams, infinite trees, and bisimulation [15].

This paper presents Dual Calculus DC$\mu\nu$ with inductive types and coinductive types. Our main results are: (1) the duality between inductive types and coinductive types with reduction, (2) strong normalization in DC$\mu\nu$, and (3) strong normalization in second-order Dual Calculus DC2.

The duality between inductive definitions and coinductive definitions is known only for definition mechanism. Inductive definition formalizes the least formula B satisfying $B \leftrightarrow A[B/X]$ where A, B are formulas, the variable X occurs only positively in A, and $A[B/X]$ is obtained from A by replacing X by B. Coinductive definition formalizes the greatest formula C satisfying $C \leftrightarrow A[C/X]$. The duality for definition mechanism between inductive definition and coinductive definition is expressed by the fact that B is equivalent to $\neg D$ where D is the greatest formula satisfying $D \leftrightarrow \neg A[D/X]$. However, its duality for reduction has not been investigated yet. We will extend this known duality from types to both terms and their reduction.

In category theory, inductive definitions are represented by initial algebras and coinductive definitions are represented by terminal coalgebras [4], and their duality is known. This paper shows the duality in a clear syntactical way by using a type system.

[9] discussed an intuitionistic sequent calculus with inductive definitions and coinductive definitions and showed its cut elimination theorem. Our cut elimination procedure is not closed in an intuitionistic fragment because we require that our cut elimination procedure respects the duality of classical logic. Our cut elimination for inductive types is the same as theirs, and on the other hand our cut elimination for coinductive types is different from theirs because of the duality.

In order for proving strong normalization, we will first present second-order Dual Calculus DC2 and show its strong normalization by interpreting it in second-order symmetric lambda-calculus [12]. Then strong normalization of DC$\mu\nu$ is proved by interpreting it in DC2 by using second-order coding of inductive and coinductive types.

Section 2 gives a definition of DC and states its duality. Section 3 introduces DC$\mu\nu$ and shows its duality. Section 4 gives examples. In section 5, we give DC2 and show its strong normalization. Section 6 proves strong normalization for DC$\mu\nu$.

2 Dual Calculus DC

This section defines Dual Calculus DC and states its duality. This system is obtained from the original Dual Calculus given in [16] by removing reduction strategies in reduction rules.

Definition 1 (Types and Expressions of DC). Let $X, Y, Z, \ldots$ range over type variables, $A, B, \ldots$ range over types, $x, y, z, \ldots$ range over variables, and $\alpha, \beta, \gamma, \ldots$ range over covariables. We assume a bijection $(-)'$ between variables and covariables, which satisfies $x'' = x$ and $\alpha'' = \alpha$. An expression (denoted by $D, E, \ldots$) is either a term (denoted by $M, N, \ldots$), a coterm (denoted by $K, L, \ldots$), or a statement (denoted by $S, T, \ldots$). We define them as follows:

Types $A ::= X \mid A \wedge A \mid A \vee A \mid \neg A$,
Expressions $D ::= M \mid K \mid S$,
Terms $M ::= x \mid \langle M, M\rangle \mid \langle M\rangle\mathtt{inl} \mid \langle M\rangle\mathtt{inr} \mid [K]\mathtt{not} \mid (S).\alpha$,
Coterms $K ::= \alpha \mid [K, K] \mid \mathtt{fst}[K] \mid \mathtt{snd}[K] \mid \mathtt{not}\langle M\rangle \mid x.(S)$,
Statements $S ::= M \bullet K$.

$(S).\alpha$ binds the covariable α in S. $x.(S)$ binds the variable x in S. We will use $_[_/_]$ for substitution. For example, the substitution $S[M/x]$ denotes the statement obtained from S by replacing x by M.

$A \wedge B$ denotes a cartesian product. $A \vee B$ denotes a disjoint sum. $\neg A$ is the type of a continuation program which returns an answer when it gets the input of type A. A variable means an ordinary variable. A covariable means an output port and gets some value after computation. A term represents an ordinary computation which becomes a value and puts values at output ports after computation. $\langle M, N\rangle$ means a pair. $\langle M\rangle\mathtt{inl}$ and $\langle M\rangle\mathtt{inr}$ mean the left injection and the right injection to a disjoint sum, respectively. When $[K]\mathtt{not}$ gets its input, it gives the input to K and computes K. $(S).\alpha$ is an abstraction of S by α. It computes S and its value is the value at the output port α. A coterm represents continuation which puts values at output ports after computation when it gets its input. $[K, L]$ gets the input of a disjoint sum. If the input is $\langle M\rangle\mathtt{inl}$, it gives M to K and computes K. If the input is $\langle M\rangle\mathtt{inr}$, it gives M to L and computes L. $\mathtt{fst}[K]$ gets the input of a cartesian product. If the input is $\langle M, N\rangle$, then it gives M to K and computes K. $\mathtt{snd}[K]$ also gets the input of a cartesian product. If the input is $\langle M, N\rangle$, then it gives N to K and computes K. $\mathtt{not}\langle M\rangle$ gets a continuation as its input. It gives M to the continuation and computes the continuation. $x.(S)$ is an abstraction of S by x. If it gets the input, it puts the input in x and computes S. $M \bullet K$ means the computation of K with the input M.

A typing judgment (denoted by J) of DC takes either the form $\Gamma \vdash \Delta \mid M : A$, the form $K : A \mid \Gamma \vdash \Delta$, or the form $\Gamma \mid S \vdash \Delta$, where Γ denotes a context $x_1 : A_1, \ldots, x_n : A_n$ that is a set of variable declarations, and Δ denotes a cocontext $\alpha_1 : B_1, \ldots, \alpha_m : B_m$ that is a set of covariable declarations. We will call M, K, and S a principal expression in those judgments.

The typing judgment $x_1 : A_1, \ldots, x_n : A_n \vdash \alpha_1 : B_1, \ldots, \alpha_m : B_m \mid M : A$ means that when each x_i has a value of type A_i, and M is computed, then M returns a value of type A or some α_i gets a value of type B_i. The judgment $K : A \mid x_1 : A_1, \ldots, x_n : A_n \vdash \alpha_1 : B_1, \ldots, \alpha_m : B_m$ means that when each x_i has a value of type A_i, some input of type A is given to K, and K is computed, then some α_i gets a value of type B_i. The judgment $x_1 : A_1, \ldots, x_n : A_n \mid S \vdash \alpha_1 : B_1, \ldots, \alpha_m : B_m$ means that when each x_i has a value of type A_i and S is computed, then some α_i gets a value of type B_i.

The typing rules are given in Figure 1. If we erase terms, coterms, statements, and the symbol |, the system becomes equivalent to the classical sequent calculus LK.

Definition 2 (Reduction). The reduction relation $\longrightarrow_{\mathtt{DC}}$ is defined as the compatible closure of the following reduction rules:

$$\frac{}{\Gamma, x : A \vdash \Delta \mid x : A}\ (AxR) \qquad \frac{}{\alpha : A \mid \Gamma \vdash \Delta, \alpha : A}\ (AxL)$$

$$\frac{\Gamma \vdash \Delta \mid M : A \quad \Gamma \vdash \Delta \mid N : B}{\Gamma \vdash \Delta \mid \langle M, N\rangle : A \wedge B}\ (\wedge R) \qquad \frac{K : A \mid \Gamma \vdash \Delta \quad L : B \mid \Gamma \vdash \Delta}{[K, L] : A \vee B \mid \Gamma \vdash \Delta}\ (\vee L)$$

$$\frac{\Gamma \vdash \Delta \mid M : A}{\Gamma \vdash \Delta \mid \langle M\rangle \mathtt{inl} : A \vee B}\ (\vee R1) \qquad \frac{K : A \mid \Gamma \vdash \Delta}{\mathtt{fst}[K] : A \wedge B \mid \Gamma \vdash \Delta}\ (\wedge L1)$$

$$\frac{\Gamma \vdash \Delta \mid M : B}{\Gamma \vdash \Delta \mid \langle M\rangle \mathtt{inr} : A \vee B}\ (\vee R2) \qquad \frac{K : B \mid \Gamma \vdash \Delta}{\mathtt{snd}[K] : A \wedge B \mid \Gamma \vdash \Delta}\ (\wedge L2)$$

$$\frac{K : A \mid \Gamma \vdash \Delta}{\Gamma \vdash \Delta \mid [K]\mathtt{not} : \neg A}\ (\neg R) \qquad \frac{\Gamma \vdash \Delta \mid M : A}{\mathtt{not}\langle M\rangle : \neg A \mid \Gamma \vdash \Delta}\ (\neg L)$$

$$\frac{\Gamma \mid S \vdash \Delta, \alpha : A}{\Gamma \vdash \Delta \mid (S).\alpha : A}\ (IR) \qquad \frac{\Gamma, x : A \mid S \vdash \Delta}{x.(S) : A \mid \Gamma \vdash \Delta}\ (IL)$$

$$\frac{\Gamma \vdash \Delta \mid M : A \quad K : A \mid \Gamma \vdash \Delta}{\Gamma \mid M \bullet K \vdash \Delta}\ (Cut)$$

Fig. 1. Typing rules of DC

$$(\beta\wedge_1)\ \langle M, N\rangle \bullet \mathtt{fst}[K] \longrightarrow_{\mathtt{DC}} M \bullet K, \qquad (\beta\vee_1)\ \langle M\rangle\mathtt{inl} \bullet [K, L] \longrightarrow_{\mathtt{DC}} M \bullet K,$$
$$(\beta\wedge_2)\ \langle M, N\rangle \bullet \mathtt{snd}[K] \longrightarrow_{\mathtt{DC}} N \bullet K, \qquad (\beta\vee_2)\ \langle M\rangle\mathtt{inr} \bullet [K, L] \longrightarrow_{\mathtt{DC}} M \bullet L,$$
$$(\beta\neg)\ [K]\mathtt{not} \bullet \mathtt{not}\langle M\rangle \longrightarrow_{\mathtt{DC}} M \bullet K,$$
$$(\beta R)\ (S).\alpha \bullet K \longrightarrow_{\mathtt{DC}} S[K/\alpha], \qquad (\beta L)\ M \bullet x.(S) \longrightarrow_{\mathtt{DC}} S[M/x],$$
$$(\eta R)\ (M \bullet \alpha).\alpha \longrightarrow_{\mathtt{DC}} M, \qquad (\eta L)\ x.(x \bullet K) \longrightarrow_{\mathtt{DC}} K,$$

where x and α are fresh in (ηL) and (ηR), respectively.

For simplicity we do not assume any reduction strategy in this system.

The type of an expression is preserved by reduction.

Proposition 1 (Subject reduction of DC). (1) *If $\Gamma \vdash \Delta \mid M : A$ and $M \longrightarrow_{\mathtt{DC}} N$, then $\Gamma \vdash \Delta \mid N : A$ holds.*

(2) *If $K : A \mid \Gamma \vdash \Delta$ and $K \longrightarrow_{\mathtt{DC}} L$, then $L : A \mid \Gamma \vdash \Delta$ holds.*

(3) *If $\Gamma \mid S \vdash \Delta$ and $S \longrightarrow_{\mathtt{DC}} T$, then $\Gamma \mid T \vdash \Delta$ holds.*

They are shown simultaneously by induction on M, K, and S.

The following duality transformation extends the duality in the sequent calculus LK to terms, coterms, and statements.

Definition 3 (Duality Transformation). The duality transformation $(-)^\circ$ from DC into itself is defined for types, terms, coterms, and statements as follows:

$$(X)^\circ = X, \qquad (\neg A)^\circ = \neg(A)^\circ,$$
$$(A \wedge B)^\circ = (A)^\circ \vee (B)^\circ, \qquad (A \vee B)^\circ = (A)^\circ \wedge (B)^\circ,$$
$$(x)^\circ = x', \qquad (\alpha)^\circ = \alpha',$$
$$(\langle M, N\rangle)^\circ = [(M)^\circ, (N)^\circ], \qquad ([K, L])^\circ = \langle (K)^\circ, (L)^\circ\rangle,$$
$$(\langle M\rangle\mathtt{inl})^\circ = \mathtt{fst}[(M)^\circ], \qquad (\mathtt{fst}[K])^\circ = \langle (K)^\circ\rangle\mathtt{inl},$$
$$(\langle M\rangle\mathtt{inr})^\circ = \mathtt{snd}[(M)^\circ], \qquad (\mathtt{snd}[K])^\circ = \langle (K)^\circ\rangle\mathtt{inr},$$
$$([K]\mathtt{not})^\circ = \mathtt{not}\langle (K)^\circ\rangle, \qquad (\mathtt{not}\langle M\rangle)^\circ = [(M)^\circ]\mathtt{not},$$
$$((S).\alpha)^\circ = \alpha'.((S)^\circ), \qquad (x.(S))^\circ = ((S)^\circ).x', \qquad (M \bullet K)^\circ = (K)^\circ \bullet (M)^\circ.$$

Note that a type and a statement are mapped to themselves. A term and a coterm are mapped to each other.

We also define transformation for judgments. If Γ is $x_1 : A_1, \ldots, x_n : A_n$, then $(\Gamma)^\circ$ is defined as $(x_1)^\circ : (A_1)^\circ, \ldots, (x_n)^\circ : (A_n)^\circ$. If Δ is $\alpha_1 : B_1, \ldots, \alpha_m : B_m$, then $(\Delta)^\circ$ is defined as $(\alpha_1)^\circ : (B_1)^\circ, \ldots, (\alpha_m)^\circ : (B_m)^\circ$. The judgment $(\Gamma \vdash \Delta \mid M : A)^\circ$ is defined as $(M)^\circ : (A)^\circ \mid (\Delta)^\circ \vdash (\Gamma)^\circ$. The judgment $(K : A \mid \Gamma \vdash \Delta)^\circ$ is defined as $(\Delta)^\circ \vdash (\Gamma)^\circ \mid (K)^\circ : (A)^\circ$. The judgment $(\Gamma \mid S \vdash \Delta)^\circ$ is defined as $(\Delta)^\circ \mid (S)^\circ \vdash (\Gamma)^\circ$.

This duality transformation is shown to preserve typing and reduction, and to be an involution. This transformation is a homomorphism for this system in the sense that it preserves typing and reduction. An important feature of DC is its duality by this transformation. A term is dual to a coterm by this homomorphism.

Proposition 2 (Duality of DC). (1) *If J is provable in DC, then $(J)^\circ$ is provable in DC.*
(2) *$D \longrightarrow_{\mathtt{DC}} E$ implies $(D)^\circ \longrightarrow_{\mathtt{DC}} (E)^\circ$.*
(3) *$((A)^\circ)^\circ = A$, $((D)^\circ)^\circ = D$, and $((J)^\circ)^\circ = J$ hold for any type A, expression D, and judgment J.*

(1) is proved by induction on the proof. (2) is proved by case analysis. (3) is proved by induction on types and expressions.

Remark. This transformation maps dual reduction rules to each other. That is, if $D \longrightarrow_{\mathtt{DC}} E$ is the reduction rules $(\beta\wedge_1)$, $(\beta\wedge_2)$, $(\beta\vee_1)$, $(\beta\vee_2)$, $(\beta\neg)$, (βR), (βL), (ηR), and (ηL), then $(D)^\circ \longrightarrow_{\mathtt{DC}} (E)^\circ$ is the reduction rules $(\beta\vee_1)$, $(\beta\vee_2)$, $(\beta\wedge_1)$, $(\beta\wedge_2)$, $(\beta\neg)$, (βL), (βR), (ηL), and (ηR), respectively.

Implication $\supset$ can be defined by $\neg$ and $\vee$ in the same way as [16].

Definition 4. We write $A \supset B$ for $\neg A \vee B$. We also write $\lambda x.M$ for $(\langle [x.(\langle M\rangle \mathtt{inr} \bullet \gamma)]\mathtt{not}\rangle \mathtt{inl} \bullet \gamma).\gamma$. We also write $N@K$ for $[\mathtt{not}\langle N\rangle, K]$.

$@$ means the application in λ-calculus. The following holds from the definition.

Proposition 3. *The following typing inference rules and reduction rule are derivable.*

$$\frac{\Gamma, x : A \vdash \Delta \mid M : B}{\Gamma \vdash \Delta \mid \lambda x.M : A \supset B}\ (\supset R) \qquad \frac{\Gamma \vdash \Delta \mid M : A \quad K : B \mid \Gamma \vdash \Delta}{M@K : A \supset B \mid \Gamma \vdash \Delta}\ (\supset L)$$

$$(\beta\supset) \quad \lambda x.M \bullet (N@K) \longrightarrow_{\mathtt{DC}} M[N/x] \bullet K$$

3 Dual Calculus DC$\mu\nu$ with Inductive and Coinductive Types

In this section, we present DC$\mu\nu$, which is an extension of DC with inductive types and coinductive types. We first extend the definition of types of DC to inductive types $\mu X.A$ and coinductive types $\nu X.A$, and then extend expressions and reduction.

Definition 5. The set of type variables is written by TyVars. We define the types of DC$\mu\nu$ (denoted by $A, B, \ldots$) as follows:

$$A ::= X \mid A \wedge A \mid A \vee A \mid \neg A \mid \mu X.A \mid \nu X.A$$

where $\mu X.A$ and $\nu X.A$ are defined when the type variable X is in $\mathrm{Pos}(A)$, and the set $\mathrm{Pos}(A)$ of *positive type variables* in the type A and the set $\mathrm{Neg}(A)$ of *negative type*

variables in the type A are defined as follows: $\mathrm{Pos}(X) = \mathtt{TyVars}$, $\mathrm{Neg}(X) = \mathtt{TyVars} \setminus \{X\}$, $\mathrm{Pos}(A_1 \wedge A_2) = \mathrm{Pos}(A_1 \vee A_2) = \mathrm{Pos}(A_1) \cap \mathrm{Pos}(A_2)$, $\mathrm{Neg}(A_1 \wedge A_2) = \mathrm{Neg}(A_1 \vee A_2) = \mathrm{Neg}(A_1) \cap \mathrm{Neg}(A_2)$, $\mathrm{Pos}(\neg B) = \mathrm{Neg}(B)$, $\mathrm{Neg}(\neg B) = \mathrm{Pos}(B)$, $\mathrm{Pos}(\mu X.B) = \mathrm{Pos}(\nu X.B) = \mathrm{Pos}(B) \cup \{X\}$, $\mathrm{Neg}(\mu X.B) = \mathrm{Neg}(\nu X.B) = \mathrm{Neg}(B) \cup \{X\}$. $\mu X.A$ and $\nu X.A$ bind X in A.

A positive type variable in A does not occur negatively in A in the usual sense. A negative type variable in A does not occur positively in A.

The inductive types and the coinductive types mean the least fixed points and the greatest fixed points respectively in the set theoretical semantics, where types are interpreted by sets, $A \wedge B$ is interpreted by the cartesian product of the sets A and B, $A \vee B$ is interpreted by the disjoint sum of the sets A and B, $\neg A$ is interpreted by the set of functions from the set A to the empty set. Let $\mathcal{P}$ be the function which maps the set V to the set $A[V/X]$. Then $\mu X.A$ and $\nu X.A$ are interpreted by the least fixed point and the greatest fixed point of the monotone function $\mathcal{P}$ respectively. Let μ be $\mu X.A$ and ν be $\nu X.A$. They will have the following properties: (a) $A[\mu/X] \subseteq \mu$, (b) $A[B/X] \subseteq B$ implies $\mu \subseteq B$, (c) $\nu \subseteq A[\nu/X]$, and (d) $B \subseteq A[B/X]$ implies $B \subseteq \nu$. Based on this meaning, we will introduce terms, coterms, and their reduction for inductive and coinductive types in the same way as [8].

Definition 6. The terms, coterms, and statements of $\mathtt{DC}\mu\nu$ are defined as follows:

$$
\begin{array}{rcl}
M & ::= & x \mid \langle M, M\rangle \mid \langle M\rangle \mathtt{inl} \mid \langle M\rangle \mathtt{inr} \mid [K]\mathtt{not} \mid (S).\alpha \mid \mathtt{in}^{\mu X.A}\langle M\rangle \mid \mathrm{coitr}^{A}_{x}\langle M, M\rangle, \\
K & ::= & \alpha \mid [K, K] \mid \mathtt{fst}[K] \mid \mathtt{snd}[K] \mid \mathtt{not}\langle M\rangle \mid x.(S) \mid \mathtt{out}^{\nu X.A}[K] \mid \mathrm{itr}^{A}_{\alpha}[K, K], \\
S & ::= & M \bullet K.
\end{array}
$$

$\mathrm{itr}^{A}_{\alpha}[K, L]$ binds α in K. $\mathrm{coitr}^{A}_{x}\langle M, N\rangle$ binds x in M.

$\mathtt{in}^{\mu X.A}\langle M\rangle$ and $\mathrm{itr}^{A}_{\alpha}[K, L]$ are the expressions for inductive types. $\mathtt{in}^{\mu X.A}$ maps an element of type $A[\mu X.A/X]$ to that of $\mu X.A$. $\mathrm{itr}^{B}_{\alpha}[K, L]$ is an iterator having the input of type $\mu X.A$ where L is a postprocessor after iteration. When it gets the input of type $\mu X.A$, first a value of type $A[\mu X.A/X]$ is computed according to the input, next a value of type $A[B/X]$ is computed by recursive invocation of the iterator, then it is given to K and K is computed to get a value of type B, and finally the value is given to L and L is computed. Dually, $\mathtt{out}^{\nu X.A}[K]$ and $\mathrm{coitr}^{A}_{x}\langle M, N\rangle$ are defined for coinductive types. $\mathtt{out}^{\nu X.A}$ maps an element of type $\nu X.A$ to that of $A[\nu X.A/X]$. When it gets the input of type $\nu X.A$, first the input is transformed into a value of type $A[\nu X.A/X]$, then the value is given to K, and finally K is computed. $\mathrm{coitr}^{B}_{x}\langle M, N\rangle$ is a coiterator of type $\nu X.A$. It transforms N of type B into a value of $\nu X.A$ according to M. Type annotations will be necessary for defining reduction rules.

Definition 7. The typing rules of $\mathtt{DC}\mu\nu$ are defined by those of DC and the following rules:

$$
\frac{\Gamma \vdash \Delta \mid M : A[\mu X.A/X]}{\Gamma \vdash \Delta \mid \mathtt{in}^{\mu X.A}\langle M\rangle : \mu X.A}\ (\mu R)
\qquad
\frac{K : A[B/X] \mid \Gamma \vdash \Delta, \alpha : B \quad L : B \mid \Gamma \vdash \Delta}{\mathrm{itr}^{B}_{\alpha}[K, L] : \mu X.A \mid \Gamma \vdash \Delta}\ (\mu L)
$$

$$
\frac{K : A[\nu X.A/X] \mid \Gamma \vdash \Delta}{\mathtt{out}^{\nu X.A}[K] : \nu X.A \mid \Gamma \vdash \Delta}\ (\nu L)
\qquad
\frac{\Gamma, x : B \vdash \Delta \mid M : A[B/X] \quad \Gamma \vdash \Delta \mid N : B}{\Gamma \vdash \Delta \mid \mathrm{coitr}^{B}_{x}\langle M, N\rangle : \nu X.A}\ (\nu R)
$$

The duality transformation can be extended from DC to $\mathtt{DC}\mu\nu$.

Definition 8 (Duality Transformation). The duality transformation for types, terms, coterms, and statements of $\mathtt{DC}\mu\nu$ is defined by those of $\mathtt{DC}$ and the following equations:

$$(\mu X.A)^\circ = \nu X.(A)^\circ, \qquad (\nu X.A)^\circ = \mu X.(A)^\circ,$$
$$(\mathtt{in}^{\mu X.A}\langle M\rangle)^\circ = \mathtt{out}^{\nu X.(A)^\circ}[(M)^\circ], \qquad (\mathtt{out}^{\nu X.A}[K])^\circ = \mathtt{in}^{\mu X.(A)^\circ}\langle (K)^\circ\rangle,$$
$$(\mathrm{itr}^A_\alpha[K,L])^\circ = \mathrm{coitr}^{(A)^\circ}_{\alpha'}\langle (K)^\circ, (L)^\circ\rangle, \qquad (\mathrm{coitr}^A_x\langle M,N\rangle)^\circ = \mathrm{itr}^{(A)^\circ}_{x'}[(M)^\circ, (N)^\circ].$$

Our reduction rules for inductive and coinductive types will be defined so that they correspond to cut elimination in the corresponding logical system LK. In the following proof figures, we will write μ, ν, and $A[B]$ for $\mu X.A$, $\nu X.A$, and $A[B/X]$ respectively. In the logical system, when the cut formula is an inductive type, cut elimination reduces the proof

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \Delta, A[\mu]}}{\Gamma \vdash \Delta, \mu}\,(\mu R) \qquad \dfrac{\dfrac{\vdots}{A[B], \Gamma \vdash \Delta, B} \quad \dfrac{\vdots}{B, \Gamma \vdash \Delta}}{\mu, \Gamma \vdash \Delta}\,(\mu L)}{\Gamma \vdash \Delta}\,(Cut)$$

to the proof

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \Delta, A[\mu]} \qquad \dfrac{\dfrac{\dfrac{\vdots}{A[B], \Gamma \vdash \Delta, B} \quad \overline{B, \Gamma \vdash \Delta, B}}{\mu, \Gamma \vdash \Delta, B}\,(\mu L) \qquad \dfrac{\dfrac{\vdots}{(A[B], \Gamma \vdash \Delta, B)} \quad \dfrac{\vdots}{(B, \Gamma \vdash \Delta)}}{A[B], A[\mu], \Gamma \vdash \Delta}\,(Cut)}{A[\mu], \Gamma \vdash \Delta}\,(Cut)}{\Gamma \vdash \Delta}\,(Cut)$$

When the cut formula is a coinductive type, cut elimination reduces a proof in a dual way to the above reduction.

When we have a function $\lambda x.M$ from A to B and the variable X is in $\mathrm{Pos}(C)$, we can define the function from $C[A/X]$ to $C[B/X]$ by extending $\lambda x.M$. We will use $\mathtt{map}^{X.C}_{A,B,x.M}\{N\}$ so that this function maps z to $\mathtt{map}^{X.C}_{A,B,x.M}\{z\}$. We will define $\mathtt{map}^{X.C}_{A,B,x.M}\{N\}$ by induction on the measure $\|C\|_X$ for a type C and a type variable X, which is defined by induction on C as follows: If X is not free in A, then $\|A\|_X = 0$. In the other cases, we assume that some X occurs in A. $\|X\|_X = 1$. $\|A \wedge B\|_X = \|A \vee B\|_X = \|A\|_X + \|B\|_X + 1$. $\|\neg A\|_X = \|A\|_X + 1$. $\|\mu Y.A\|_X = \|\nu Y.A\|_X = \|A\|_X + \|A\|_Y + 1$.

Note that if X is not free in B and we have $X \neq Y$, then $\| A \|_X = \| A[B/Y] \|_X$.

Definition 9. Assume a type variable X and types A, B, C are given and X is is not free in A and B. For a variable x and terms M and N, we define the term $\mathtt{map}^{X.C}_{A,B,x.M}\{N\}$ by induction on $\| C \|_X$ as follows:

$$\mathtt{map}^{X.X}_{A,B,x.M}\{N\} = (N \bullet x.(M \bullet \alpha)).\alpha,$$
$$\mathtt{map}^{X.C}_{A,B,x.M}\{N\} = N \qquad (X \text{ does not occur in } C),$$
$$\mathtt{map}^{X.C\wedge D}_{A,B,x.M}\{N\} = \langle\, \mathtt{map}^{X.C}_{A,B,x.M}\{(N \bullet \mathtt{fst}[\alpha]).\alpha\},\, \mathtt{map}^{X.D}_{A,B,x.M}\{(N \bullet \mathtt{snd}[\beta]).\beta\}\,\rangle,$$
$$\mathtt{map}^{X.C\vee D}_{A,B,x.M}\{N\} = (N \bullet [\, y.(\langle \mathtt{map}^{X.C}_{A,B,x.M}\{y\}\,\rangle \mathtt{inl} \bullet \gamma), z.(\langle \mathtt{map}^{X.D}_{A,B,x.M}\{z\}\,\rangle \mathtt{inr} \bullet \gamma)\,]).\gamma,$$
$$\mathtt{map}^{X.\neg C}_{A,B,x.M}\{N\} = [\, z.(N \bullet \mathtt{not}\langle\, \mathtt{map}^{X.C}_{B,A,x.M}\{z\}\,\rangle\,)\,]\mathtt{not},$$
$$\mathtt{map}^{X.\mu Y.C}_{A,B,x.M}\{N\} = (\, N \bullet \mathrm{itr}^{\mu_B}_\alpha[\, z.(\, \mathtt{in}^{\mu_B}\langle\, \mathtt{map}^{X.C[\mu_B/Y]}_{A,B,x.M}\{\, z\,\}\,\rangle \bullet \alpha\,),\, \beta\,]).\beta,$$
$$\mathtt{map}^{X.\nu Y.C}_{A,B,x.M}\{N\} = \mathrm{coitr}^{\nu_A}_z\langle\, \mathtt{map}^{X.C[\nu_A/Y]}_{A,B,x.M}\{\,(z \bullet \mathtt{out}^{\nu_A}[\alpha]).\alpha\,\},\, N\,\rangle,$$

where we write μ_B and ν_A for $\mu Y.C[B/X]$ and $\nu Y.C[A/X]$, respectively. For a covariable α and coterms K and L, we also define

$$\mathtt{map}^{X.C}_{A,B,\alpha.K}\{L\} = (\mathtt{map}^{X.(C)^\circ}_{(B)^\circ,(A)^\circ,\alpha'.(K)^\circ}\{(L)^\circ\})^\circ.$$

Note that $\|\mu Y.C\|_X > \|C[\mu_B/Y]\|_X$ and $\|\mu Y.C\|_X > \|C[\nu_A/Y]\|_X$ hold since X is not free in μ_B and ν_A. We cannot replace $C[\mu_B/Y]$ by C in the definition of $\mathtt{map}^{X.\mu Y.C}_{A,B,x.M}\{N\}$ because of the type annotation for $\mathtt{in}$. For readability, we sometimes write $\mathtt{map}^{X.C}_{A,B,x}\{M,N\}$ and $\mathtt{map}^{X.C}_{A,B,\alpha}\{K,L\}$ for $\mathtt{map}^{X.C}_{A,B,x.M}\{N\}$ and $\mathtt{map}^{X.C}_{A,B,\alpha.K}\{L\}$, respectively.

Proposition 4. *Assume X is in* Pos(C) *and* Neg(D)*. The following are derivable:*

$$\frac{\Gamma, x : A \vdash \Delta \mid M : B}{\Gamma, z : C[A/X] \vdash \Delta \mid \mathtt{map}^{X.C}_{A,B,x.M}\{z\} : C[B/X]} \qquad \frac{\Gamma, x : B \vdash \Delta \mid M : A}{\Gamma, z : D[A/X] \vdash \Delta \mid \mathtt{map}^{X.D}_{A,B,x.M}\{z\} : D[B/X]}$$

$$\frac{K\colon A \mid \Gamma \vdash \Delta, \alpha\colon B}{\mathtt{map}^{X.C}_{A,B,\alpha.K}\{\beta\}\colon C[A/X] \mid \Gamma \vdash \Delta, \beta\colon C[B/X]} \qquad \frac{K\colon B \mid \Gamma \vdash \Delta, \alpha\colon A}{\mathtt{map}^{X.D}_{A,B,\alpha.K}\{\beta\}\colon D[A/X] \mid \Gamma \vdash \Delta, \beta\colon D[B/X]}$$

They are proved by induction on $\|C\|_X$ and $\|D\|_X$.

Definition 10. We define the one-step reduction relation $\longrightarrow_{\mathtt{DC}\mu\nu}$ of $\mathtt{DC}\mu\nu$ as the compatible closure of the reduction rules of $\mathtt{DC}$ and the following reduction rules:

$$(\beta\mu) \quad \mathtt{in}^{\mu X.C}\langle M\rangle \bullet \mathrm{itr}^A_\alpha[K,L] \longrightarrow_{\mathtt{DC}\mu\nu} M \bullet \mathtt{map}^{X.C}_{\mu X.C,A,\beta}\{\,\mathrm{itr}^A_\alpha[K,\beta],\ K[L/\alpha]\,\},$$
$$(\beta\nu) \quad \mathrm{coitr}^A_x\langle M,N\rangle \bullet \mathtt{out}^{\nu X.C}[K] \longrightarrow_{\mathtt{DC}\mu\nu} \mathtt{map}^{X.C}_{A,\nu X.C,z}\{\,\mathrm{coitr}^A_x\langle M,z\rangle,\ M[N/x]\,\} \bullet K.$$

This system has subject reduction. That is, when we replace $\mathtt{DC}$ by $\mathtt{DC}\mu\nu$ in the statement of Proposition 1, the statement holds also for $\mathtt{DC}\mu\nu$.

The following duality transformation $(-)^\circ$ preserves typing and reduction, and is an involution in $\mathtt{DC}\mu\nu$. This is proved similarly to Proposition 2.

Theorem 1 (Duality of $\mathtt{DC}\mu\nu$). (1) *If J is provable in $\mathtt{DC}\mu\nu$, then $(J)^\circ$ is provable in $\mathtt{DC}\mu\nu$.*

(2) *$D \longrightarrow_{\mathtt{DC}\mu\nu} E$ implies $(D)^\circ \longrightarrow_{\mathtt{DC}\mu\nu} (E)^\circ$ for any expressions D and E.*

(3) *$((A)^\circ)^\circ = A$, $((D)^\circ)^\circ = D$, and $((J)^\circ)^\circ = J$ hold for any type A, expression D, and judgment J of $\mathtt{DC}\mu\nu$.*

Proposition 5. *If $D \longrightarrow_{\mathtt{DC}\mu\nu} E$ is the rules $(\beta\mu)$ and $(\beta\nu)$, then $(D)^\circ \longrightarrow_{\mathtt{DC}\mu\nu} (E)^\circ$ is $(\beta\nu)$ and $(\beta\mu)$ respectively.*

We have shown the duality of inductive types and coinductive types. Theorem 1 shows that the duality transformation is a homomorphic involution. The description of a type is defined as the set of the type itself, its terms, its coterms, and their reduction. The duality transformation maps the description of an inductive type and that of a coinductive type to each other. That is, (1) Definition 8 shows that the inductive type $\mu X.A$ is mapped to the coinductive type $\nu X.(A)^\circ$, the term constructed by $\mathtt{in}$ for the inductive type is mapped to the coterm constructed by $\mathtt{out}$ for the coinductive type, and the coterm constructed by itr for the inductive type is mapped to the term constructed by coitr for the coinductive type. (2) Proposition 5 shows that the cut elimination of the inductive type is mapped to the cut elimination of the coinductive type, (3) the coinductive type is mapped to the inductive type in a similar way to (1) and (2).

Remark. We cannot define our typing system by using

$$\frac{K : C[A/X] \mid \Gamma \vdash \Delta, \alpha : A}{\mathrm{itr}^A_\alpha[K,\beta] : \mu X.C \mid \Gamma \vdash \Delta, \beta : A}\ (\mu L')$$

instead of the typing rule (μL). If we used ($\mu L'$), the set of terms would not be closed under substitution, because $\mathrm{itr}^A_\alpha[K,L]$ would not have typing rules for it and hence it would not be a term, though it is obtained from $\mathrm{itr}^A_\alpha[K,\beta]$ by substituting L for β.

4 Examples

In this section we show some examples of inductive and coinductive types in $\mathrm{DC}\mu\nu$.

Let X_0 be a distinguished type variable. We use the following abbreviations: $\top = \neg X_0 \vee X_0$, $\quad \bot = \neg X_0 \wedge X_0$, and $* = \lambda x.x$.

The type $\mathtt{Nat}$ of natural numbers can be represented by:

$$\mathtt{Nat} = \mu X.(\top \vee X), \qquad \mathtt{0} = \mathtt{in}^{\mathtt{Nat}}\langle\, (*)\mathtt{inl}\,\rangle, \qquad \mathtt{succ}\langle M\rangle = \mathtt{in}^{\mathtt{Nat}}\langle\, \langle M\rangle \mathtt{inr}\,\rangle,$$

where $\mathtt{0}$ is the zero and $\mathtt{succ}$ is the successor. We can prove $\Gamma \vdash \Delta \mid \mathtt{0} : \mathtt{Nat}$. We can also prove $\Gamma \vdash \Delta \mid \mathtt{succ}\langle M\rangle : \mathtt{Nat}$ from $\Gamma \vdash \Delta \mid M : \mathtt{Nat}$. The n-th natural number $\tilde{n}$ is represented by $\mathtt{succ}\langle\mathtt{succ}\langle \ldots \mathtt{succ}\langle \mathtt{0}\rangle \ldots\rangle\rangle$ (n times of $\mathtt{succ}$). We will write $M[_/x]^n(N)$ for $M[M[\ldots[M[N/x]/x]\ldots/x]/x]$ (n times of M). We define a coterm $\mathtt{Itr}^B[F,N,K]$ of type $\mathtt{Nat}$ by $\mathrm{itr}^B_\alpha[\,[y.(N \bullet \alpha), x.(F \bullet (x@\alpha))], K\,]$, where y is not free in N, F has type $B \supset B$, and N and K are of type B. When the coterm $\mathtt{Itr}^B[F,N,K]$ gets $\tilde{n}$ as its input, it computes n-time iterations of applying the function F to N, and passes the output to K. We reduce $\tilde{n} \bullet \mathtt{Itr}^B[\lambda x.M, N, K]$ to $M[_/x]^n(N) \bullet K$.

The type $\mathtt{List}(A)$ of lists of elements of type A is represented by:

$$\mathtt{List}(A) = \mu X.(\top \vee (A \wedge X)),$$
$$\mathtt{nil} = \mathtt{in}^{\mathtt{List}(A)}\langle\, (*)\mathtt{inl}\,\rangle, \qquad \mathtt{cons}\langle M, Nl\rangle = \mathtt{in}^{\mathtt{List}(A)}\langle\, \langle\, \langle M, Nl\rangle\,\rangle \mathtt{inr}\,\rangle.$$

$\mathtt{nil}$ is the empty list and $\mathtt{cons}$ is the list constructor. In $\mathrm{DC}\mu\nu$, $\Gamma \vdash \Delta \mid \mathtt{nil} : \mathtt{List}(A)$ is provable. $\Gamma \vdash \Delta \mid \mathtt{cons}\langle M, Nl\rangle : \mathtt{List}(A)$ is also provable from $\Gamma \vdash \Delta \mid M : A$ and $\Gamma \vdash \Delta \mid Nl : \mathtt{List}(A)$.

We can also define the type $\mathtt{Stream}(A)$ of streams of elements of type A by:

$$\mathtt{Stream}(A) = \nu X.(A \wedge X),$$
$$\mathtt{cons}\langle M, Ns\rangle = \mathrm{coitr}^{A \wedge \mathtt{Stream}(A)}_x\langle\, \langle \pi_1(x), (\pi_2(x) \bullet \mathtt{out}^{\mathtt{Stream}(A)}[\alpha]).\alpha\rangle, \langle M, Ns\rangle\,\rangle,$$
$$\mathtt{hd}[K] = \mathtt{out}^{\mathtt{Stream}(A)}[\,\mathtt{fst}[K]\,], \qquad \mathtt{tl}[L] = \mathtt{out}^{\mathtt{Stream}(A)}[\,\mathtt{snd}[L]\,],$$

where $\pi_1(M)$ is the first projection of M defined by $(M \bullet \mathtt{fst}[\alpha]).\alpha$, and $\pi_2(M)$ is the second projection of M defined by $(M \bullet \mathtt{snd}[\alpha]).\alpha$. The term $\mathtt{cons}\langle M, Ns\rangle$ constructs a new stream from a given element M and a given stream Ns. The coterm $\mathtt{hd}[K]$ receives the first element from a given stream and gives it to K. The coterm $\mathtt{tl}[L]$ removes the first element from a given stream and gives the resulting stream to L. We can prove $\Gamma \vdash \Delta \mid \mathtt{cons}\langle M, Ns\rangle : \mathtt{Stream}(A)$ from $\Gamma \vdash \Delta \mid M : A$ and $\Gamma \vdash \Delta \mid Ns : \mathtt{Stream}(A)$. We can also prove $\mathtt{hd}[K] : \mathtt{Stream}(A) \mid \Gamma \vdash \Delta$ from $K : A \mid \Gamma \vdash \Delta$. We can also prove $\mathtt{tl}[L] : \mathtt{Stream}(A) \mid \Gamma \vdash \Delta$ from $L : \mathtt{Stream}(A) \mid \Gamma \vdash \Delta$. We reduce $\mathtt{cons}\langle M, Ns\rangle \bullet \mathtt{hd}[K]$ to $M \bullet K$. We also reduce $\mathtt{cons}\langle M, Ns\rangle \bullet \mathtt{tl}^{n+1}[\mathtt{hd}[K]]$ to $Ns \bullet \mathtt{tl}^n[\mathtt{hd}[K]]$, where $\mathtt{tl}^n[\mathtt{hd}[K]]$ is defined by $\mathtt{tl}[\mathtt{tl}[\ldots\mathtt{tl}[\,\mathtt{hd}[K]\,]\ldots]]$ (n times of $\mathtt{tl}$).

Proposition 6. $\mathtt{Nat}$ *is dual to* $\mathtt{Stream}(\bot)$*, that is,* $(\mathtt{Nat})^\circ = \mathtt{Stream}(\bot)$*,* $(\mathtt{0})^\circ = \mathtt{hd}[(*)^\circ]$*, and* $(\mathtt{succ}\langle M\rangle)^\circ = \mathtt{tl}[(M)^\circ]$ *hold.*

We can understand this duality as follows. $\top$ means the singleton set $\{*\}$. $\bot$ means the type of a program that returns some answer after computation with the input $*$ since $\bot$ is equivalent to $\neg\top$. $\mathtt{Nat}$ means the infinite disjoint sum $\top+\top+\top+\ldots$. $\mathtt{Stream}(\bot)$ means the infinite cartesian product $\bot\times\bot\times\bot\times\ldots$. Since a term in $\mathtt{Stream}(\bot)$ is equivalent to a coterm in $\mathtt{Nat}$, when the term gets some natural number and is computed, it returns some answer. When the term gets the natural number $\tilde{n}$, since $\tilde{n}$ is $*$ in the n-th $\top$ in $\top+\top+\ldots$, the term in the n-th $\bot$ in $\bot\times\bot\times\ldots$ is given the input $*$ and it is computed to give some answer.

By using the duality of $\mathtt{DC}\mu\nu$, inductive types can be defined by using coinductive types, and coinductive types are defined by inductive types. That is, $\mu X.A$ is equivalent to $\neg\nu X.(A)^\circ$ and $\nu X.A$ is equivalent to $\neg\nu X.(A)^\circ$. This is because the behavior of term M and coterm K of type A are simulated by the term $[(M)^\circ]\mathtt{not}$ and coterm $\mathtt{not}\langle(K)^\circ\rangle$ of type $\neg(A)^\circ$ respectively. For example, the type $\mathtt{Nat}'$ of natural numbers and the type $\mathtt{Stream}'(A)$ of streams can be also defined as $\mathtt{Nat}' = \neg\nu X.(\bot\wedge X)$ and $\mathtt{Stream}'(A) = \neg\mu X.((A)^\circ\vee X)$ respectively.

5 Second-Order Dual Calculus DC2

We introduce a second-order extension DC2 of DC. We will prove its strong normalization by interpreting it in the second-order symmetric λ-calculus.

Definition 11. An expression is defined to be strongly normalizing if there does not exist any infinite reduction sequence starting from the expression.

First, we define a second-order extension DC2 of DC.

Definition 12 (DC2). The types, terms, coterms, and statements of DC2 are defined by:
Types $A ::= X \mid A\wedge A \mid A\vee A \mid \neg A \mid \forall X.A \mid \exists X.A$,
Terms $M ::= x \mid \langle M, M\rangle \mid \langle M\rangle\mathtt{inl} \mid \langle M\rangle\mathtt{inr} \mid [K]\mathtt{not} \mid (S).\alpha \mid \langle M\rangle\mathtt{a} \mid \langle M\rangle\mathtt{e}$,
Coterms $K ::= \alpha \mid [K, K] \mid \mathtt{fst}[K] \mid \mathtt{snd}[K] \mid \mathtt{not}\langle M\rangle \mid x.(S) \mid \mathtt{a}[K] \mid \mathtt{e}[K]$,
Statements $S ::= M \bullet K$.
The typing rules and reduction rules (denoted by $\longrightarrow_{\mathtt{DC2}}$) of DC2 are defined by extending the rules of DC with the following rules:

$$\frac{\Gamma\vdash\Delta \mid M : A}{\Gamma\vdash\Delta \mid \langle M\rangle\mathtt{a} : \forall Z.A}\ (\forall R) \qquad \frac{K : A[B/X] \mid \Gamma\vdash\Delta}{\mathtt{a}[K] : \forall X.A \mid \Gamma\vdash\Delta}\ (\forall L)$$

$$\frac{\Gamma\vdash\Delta \mid M : A[B/X]}{\Gamma\vdash\Delta \mid \langle M\rangle\mathtt{e} : \exists X.A}\ (\exists R) \qquad \frac{K : A \mid \Gamma\vdash\Delta}{\mathtt{e}[K] : \exists Z.A \mid \Gamma\vdash\Delta}\ (\exists L)$$

$$(\beta\forall)\quad \langle M\rangle\mathtt{a} \bullet \mathtt{a}[K] \longrightarrow_{\mathtt{DC2}} M \bullet K, \qquad (\beta\exists)\quad \langle M\rangle\mathtt{e} \bullet \mathtt{e}[K] \longrightarrow_{\mathtt{DC2}} M \bullet K,$$

where Z is not free in Γ and Δ in $(\forall R)$ and $(\exists L)$. We write $\longrightarrow^{+}_{\mathtt{DC2}}$ to denote the transitive closure of $\longrightarrow_{\mathtt{DC2}}$.

We have the new constructors $\mathtt{a}$ and $\mathtt{e}$, which are trivial witnesses for the quantifiers at the level of expressions, so that the system has subject reduction.

This system has subject reduction. That is, when we replace DC by DC2 in the statement of Proposition 1, the statement holds also for DC2.

Remark. If we did not have those new constructors, the subject reduction would fail. If we chose the following $(\forall R')$ and $(\forall L')$ instead of $(\forall R)$ and $(\forall L)$,

$$\frac{\Gamma \vdash \Delta \mid M : A}{\Gamma \vdash \Delta \mid M : \forall Z.A}\ (\forall R') \qquad \frac{K : A[B/X] \mid \Gamma \vdash \Delta}{K : \forall X.A \mid \Gamma \vdash \Delta}\ (\forall L')$$

then the following would be a counter-example for subject reduction: we would have $\Gamma \mid (x \bullet \mathtt{fst}[\alpha]).\alpha \bullet \beta \vdash \Delta$ where Γ is $x\colon X \wedge Y$, Δ is $\beta\colon \forall Z.X$, and $Z \neq X, Y$, but would not have $\Gamma \mid x \bullet \mathtt{fst}[\beta] \vdash \Delta$, though $(x \bullet \mathtt{fst}[\alpha]).\alpha \bullet \beta$ is reduced to $x \bullet \mathtt{fst}[\beta]$.

In other systems such as $\lambda\mu$-calculus [11], the constructor a is not necessary for subject reduction while the constructor e is necessary for it. In our system, since $\forall$ and $\exists$ are dual, the constructor a is also necessary for subject reduction.

Next we give a definition of the second-order symmetric λ-calculus $S\lambda 2$ [12]. The symmetric λ-calculus is introduced by Barbanera and Berardi [1] as a classical extension of the λ-calculus. Strong normalization of its second-order extension $S\lambda 2$ is proved by Parigot [12].

Definition 13 **($S\lambda 2$).** We define the second-order symmetric λ-calculus $S\lambda 2$. The types of $S\lambda 2$ are either the special type $\bot$ or m-types (denoted by $\tau, \sigma, \ldots$) given by:

$$\tau ::= X \mid X^{\bot} \mid \tau \times \tau \mid \tau + \tau \mid \forall X.\tau \mid \exists X.\tau$$

where $X, Y, \ldots$ range over type variables. $\forall X.\tau$ and $\exists X.\tau$ bind X in τ. The negation $(\tau)^{\bot}$ of τ is defined by: $(X)^{\bot} = X^{\bot}$, $(\tau \times \sigma)^{\bot} = (\tau)^{\bot} + (\sigma)^{\bot}$, $(\forall X.\tau)^{\bot} = \exists X.(\tau)^{\bot}$, $(X^{\bot})^{\bot} = X$, $(\tau + \sigma)^{\bot} = (\tau)^{\bot} \times (\sigma)^{\bot}$, $(\exists X.\tau)^{\bot} = \forall X.(\tau)^{\bot}$.

$x, y, \ldots, \alpha, \beta, \ldots$ range over variables. The terms of $S\lambda 2$, denoted by $t, u, \ldots$, are defined by $t ::= x \mid \mathtt{inj}_1(t) \mid \mathtt{inj}_2(t) \mid \langle t, t\rangle \mid t * t \mid \lambda x.t \mid \mathtt{a}(t) \mid \mathtt{e}(t)$.

The one-step reduction relation $\longrightarrow_{S\lambda 2}$ of $S\lambda 2$ is defined as the compatible closure of the following rules:

$$\begin{array}{llll}
(\beta_r) & (\lambda x.t) * u \longrightarrow_{S\lambda 2} t[u/x], & (\beta_l) & u * (\lambda x.t) \longrightarrow_{S\lambda 2} t[u/x], \\
(\beta_{\times+}1) & \langle t_1, t_2\rangle * \mathtt{inj}_1(u) \longrightarrow_{S\lambda 2} t_1 * u, & (\beta_{+\times}1) & \mathtt{inj}_1(u) * \langle t_1, t_2\rangle \longrightarrow_{S\lambda 2} u * t_1, \\
(\beta_{\times+}2) & \langle t_1, t_2\rangle * \mathtt{inj}_2(u) \longrightarrow_{S\lambda 2} t_2 * u, & (\beta_{+\times}2) & \mathtt{inj}_2(u) * \langle t_1, t_2\rangle \longrightarrow_{S\lambda 2} u * t_2, \\
(\beta_{\forall\exists}) & \mathtt{a}(t) * \mathtt{e}(u) \longrightarrow_{S\lambda 2} t * u, & (\beta_{\exists\forall}) & \mathtt{e}(u) * \mathtt{a}(t) \longrightarrow_{S\lambda 2} u * t, \\
(\eta_r) & \lambda y.(y * t) \longrightarrow_{S\lambda 2} t, & (\eta_l) & \lambda y.(t * y) \longrightarrow_{S\lambda 2} t,
\end{array}$$

where y is not free in t in (η_l) and (η_r).

A typing context (denoted by Γ, Δ) is a finite set and of the form $x_1 : \tau_1, \ldots, x_n : \tau_n$. A judgment of $S\lambda 2$ takes either the form $\Gamma \vdash t : \tau$ or $\Gamma \vdash t : \bot$. The typing rules of $S\lambda 2$ is defined as follows:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ (\mathrm{Ax}) \qquad \frac{\Gamma, x : \tau \vdash t : \bot}{\Gamma \vdash \lambda x.t : (\tau)^{\bot}}\ (\mathrm{abs}) \qquad \frac{\Gamma \vdash t : (\tau)^{\bot} \quad \Gamma \vdash u : \tau}{\Gamma \vdash t * u : \bot}\ (\mathrm{app})$$

$$\frac{\Gamma \vdash t : \tau_i}{\Gamma \vdash \mathtt{inj}_i(t) : \tau_1 + \tau_2}\ (+_i)\ (i = 1, 2) \qquad \frac{\Gamma \vdash t : \tau \quad \Gamma \vdash u : \sigma}{\Gamma \vdash \langle t, u\rangle : \tau \times \sigma}\ (\times)$$

$$\frac{\Gamma \vdash t : \tau}{\Gamma \vdash \mathtt{a}(t) : \forall X.\tau}\ (\forall)\ (X \text{ is not free in } \Gamma) \qquad \frac{\Gamma \vdash t : \tau[\sigma/X]}{\Gamma \vdash \mathtt{e}(t) : \exists X.\tau}\ (\exists)$$

Theorem 2 (Strong normalization of $S\lambda 2$ [12]). *Every typable term is strongly normalizing in $S\lambda 2$.*

The following translation maps DC2 to $S\lambda 2$.

Definition 14. Let A be a type of DC2. The type $(A)^\dagger$ of $S\lambda 2$ is defined as follows:

$$(X)^\dagger = X, \qquad (A \wedge B)^\dagger = (A)^\dagger \times (B)^\dagger, \qquad (\forall X.A)^\dagger = \forall X.(A)^\dagger,$$
$$(\neg A)^\dagger = ((A)^\dagger)^\bot, \qquad (A \vee B)^\dagger = (A)^\dagger + (B)^\dagger, \qquad (\exists X.A)^\dagger = \exists X.(A)^\dagger.$$

Let D be an expression of DC2. The term $(D)^\dagger$ of $S\lambda 2$ is defined by:

$$(x)^\dagger = x, \qquad ((S).\alpha)^\dagger = \lambda\alpha.(S)^\dagger, \qquad (\alpha)^\dagger = \alpha, \qquad (x.(S))^\dagger = \lambda x.(S)^\dagger,$$
$$(\langle M\rangle \mathtt{a})^\dagger = \mathtt{a}((M)^\dagger), \qquad (\langle M\rangle \mathtt{e})^\dagger = \mathtt{e}((M)^\dagger),$$
$$(\mathtt{e}[K])^\dagger = \mathtt{a}((K)^\dagger), \qquad (\mathtt{a}[K])^\dagger = \mathtt{e}((K)^\dagger),$$
$$(\langle M\rangle \mathtt{inl})^\dagger = \mathtt{inj}_1((M)^\dagger), \qquad (\mathtt{fst}[K])^\dagger = \mathtt{inj}_1((K)^\dagger),$$
$$(\langle M\rangle \mathtt{inr})^\dagger = \mathtt{inj}_2((M)^\dagger), \qquad (\mathtt{snd}[K])^\dagger = \mathtt{inj}_2((K)^\dagger),$$
$$(\langle M, N\rangle)^\dagger = \langle (M)^\dagger, (N)^\dagger\rangle, \qquad ([K, L])^\dagger = \langle (K)^\dagger, (L)^\dagger\rangle,$$
$$([K]\mathtt{not})^\dagger = (K)^\dagger, \qquad (\mathtt{not}\langle M\rangle)^\dagger = \lambda x.((M)^\dagger * x), \qquad (M \bullet K)^\dagger = (M)^\dagger * (K)^\dagger.$$

We define the translation of $\mathtt{not}\langle M\rangle$ by using η-expansion, so that all reductions in DC2 are simulated in $S\lambda 2$.

$(\Gamma)^\dagger$ and $((\Delta)^\dagger)^\bot$ are defined as $x_1 : (A_1)^\dagger, \ldots, x_n : (A_n)^\dagger$ and $\alpha_1 : ((B_1)^\dagger)^\bot, \ldots, \alpha_m : ((B_m)^\dagger)^\bot$ respectively if Γ is $x_1 : A_1, \ldots, x_n : A_n$, and Δ is $\alpha_1 : B_1, \ldots, \alpha_m : B_m$. For a judgment J of DC2, the judgment $(J)^\dagger$ of $S\lambda 2$ is defined as follows: The judgment $(\Gamma \vdash \Delta \mid M : A)^\dagger$ is defined as $(\Gamma)^\dagger, ((\Delta)^\dagger)^\bot \vdash (M)^\dagger : (A)^\dagger$. The judgment $(K : A \mid \Gamma \vdash \Delta)^\dagger$ is defined as $(\Gamma)^\dagger, ((\Delta)^\dagger)^\bot \vdash (K)^\dagger : ((A)^\dagger)^\bot$. The judgment $(\Gamma \mid S \vdash \Delta)^\dagger$ is defined as $(\Gamma)^\dagger, ((\Delta)^\dagger)^\bot \vdash (S)^\dagger : \bot$.

This translation preserves provability and one-step reductions.

Proposition 7. (1) *If J is provable in* DC2*, then $(J)^\dagger$ is provable in $S\lambda 2$.*
(2) *$D \longrightarrow_{\mathtt{DC2}} E$ implies $(D)^\dagger \longrightarrow_{S\lambda 2} (E)^\dagger$.*

Proof. (1) is shown by induction on the proof of J. (2) is shown by induction on the definition of $\longrightarrow_{\mathtt{DC2}}$. □

We can obtain strong normalization of DC2 from the above proposition.

Theorem 3 (Strong normalization of DC2). *Every typable expression is strongly normalizing in* DC2*.*

Proof. Assume there is an infinite reduction sequence $D = D_0 \longrightarrow_{\mathtt{DC2}} D_1 \longrightarrow_{\mathtt{DC2}} \ldots$ starting from D. From Proposition 7, $(D)^\dagger$ is typable in $S\lambda 2$, and $(D)^\dagger \longrightarrow_{S\lambda 2} (D_1)^\dagger \longrightarrow_{S\lambda 2} \ldots$ is an infinite reduction sequence. This contradicts Theorem 2. □

6 Strong Normalization of DC$\mu\nu$

In this section, we prove strong normalization in DC$\mu\nu$. We will give the second-order encoding of DC$\mu\nu$ in DC2, and show that one-step reduction in DC$\mu\nu$ is translated to one or more step reduction in DC2.

We use the following degree of expressions in DC$\mu\nu$ for defining the second-order coding of inductive and coinductive types.

Definition 15. Let D be an expression in $\mathtt{DC}\mu\nu$. $\|D\|$ is defined by: $\|x\| = \|\alpha\| = 0$, $\|\langle M,N\rangle\| = \|\mathrm{coitr}^A_x\langle M,N\rangle\| = \mathtt{max}(\|M\|,\|N\|)$, $\|M \bullet K\| = \mathtt{max}(\|M\|,\|K\|)$, $\|[K,L]\| = \|\mathrm{itr}^A_\alpha[K,L]\| = \mathtt{max}(\|K\|,\|L\|)$, $\|(S).\alpha\| = \|x.(S)\| = \|S\|$, $\|\langle M\rangle\mathtt{inl}\| = \|\langle M\rangle\mathtt{inr}\| = \|\mathtt{not}\langle M\rangle\| = \|M\|$, $\|\mathtt{in}^{\mu X.A}\langle M\rangle\| = \|M\| + \|A\|_X + 1$, $\|\mathtt{fst}[K]\| = \|\mathtt{snd}[K]\| = \|[K]\mathtt{not}\| = \|K\|$, $\|\mathtt{out}^{\nu X.A}[K]\| = \|K\| + \|A\|_X + 1$.

$|D|$ is defined by: $|x| = |\alpha| = 0$, $|\langle M,N\rangle| = |\mathrm{coitr}^A_x\langle M,N\rangle| = |M|+|N|+1$, $|M\bullet K| = |M| + |K| + 1$, $|[K,L]| = |\mathrm{itr}^A_\alpha[K,L]| = |K| + |L| + 1$, $|(S).\alpha| = |x.(S)| = |S| + 1$, $|\langle M\rangle\mathtt{inl}| = |\langle M\rangle\mathtt{inr}| = |\mathtt{not}\langle M\rangle| = |\mathtt{in}^{\mu X.A}\langle M\rangle| = |M| + 1$, $|\mathtt{fst}[K]| = |\mathtt{snd}[K]| = |[K]\mathtt{not}| = |\mathtt{out}^{\nu X.A}[K]| = |K| + 1$.

The degree $\mathtt{deg}(D)$ of the expression D is defined as the triple $(\|D\|, |D|, n)$ where $n = 1$ if D is of the form $\mathrm{coitr}^A_x\langle M,N\rangle$ or $\mathtt{out}^{\nu X.A}[K]$, and otherwise $n = 0$. We also define the order of degrees by the lexicographic order.

$|D|$ is the number of constructors in the expression D. $\|D\|$ is the maximum summation of $(\|A\|_X + 1)$ for $\mathtt{in}^{\mu X.A}\langle M\rangle$ and $\mathtt{out}^{\nu X.A}[K]$ in paths in D. We have $\|E\| \le \|D\|$ and $|E| < |D|$ when the expression E is a proper subexpression of D. The degree satisfies the following properties.

Lemma 1. (1) $\|D\| = \|(D)^\circ\|$ *and* $|D| = |(D)^\circ|$ *hold.*
(2) $\|\,\mathtt{map}^{X.A}_{B,C,x}\{M,N\}\,\| \le \|M\| + \|N\| + \|A\|_X$ *holds.*
(3) $\mathtt{deg}(\mathtt{in}^{\mu X.A}\langle M\rangle) > \mathtt{deg}(\,\mathtt{map}^{X.A}_{\mu X.A,Y,\alpha}\{\,x.(y \bullet (x@\alpha)),\beta\,\})$ *holds.*

Proof. (1) They are shown by induction on D. (2) The claim is shown by induction on $\|A\|_X$. (3) The claim is proved by using (2). □

We present the second-order encoding for $\mathtt{DC}\mu\nu$. We will write $\lambda(x,\alpha).S$ for $\lambda x.((S).\alpha)$. Then $(\lambda(x,\alpha).S) \bullet (N@K)$ is reduced to $S[N/x][K/\alpha]$.

Definition 16 (Translation $\overline{(-)}$ from $\mathtt{DC}\mu\nu$ into $\mathtt{DC2}$). Let A be a type of $\mathtt{DC}\mu\nu$. The type $\overline{A}$ of DC2 is defined as follows:

$$\overline{X} = X, \quad \overline{A \wedge B} = \overline{A} \wedge \overline{B}, \quad \overline{\mu X.A} = \forall X.((\overline{A} \supset X) \supset X),$$
$$\overline{\neg A} = \neg\overline{A}, \quad \overline{A \vee B} = \overline{A} \vee \overline{B}, \quad \overline{\nu X.A} = \exists X.(\neg(\neg\overline{A} \wedge X) \wedge X),$$

where $\supset$ is defined in Definition 4. For an expression D of $\mathtt{DC}\mu\nu$, the expression $\overline{D}$ of DC2 is defined by induction on $\mathtt{deg}(D)$ as follows:

$$\overline{x} = x, \quad \overline{\alpha} = \alpha, \quad \overline{(S).\alpha} = (\overline{S}).\alpha, \quad \overline{x.(S)} = x.(\overline{S}),$$
$$\overline{\langle M,N\rangle} = \langle\overline{M},\overline{N}\rangle, \quad \overline{[K,L]} = [\overline{K},\overline{L}],$$
$$\overline{\langle M\rangle\mathtt{inl}} = \langle\overline{M}\rangle\mathtt{inl}, \quad \overline{\mathtt{fst}[K]} = \mathtt{fst}[\overline{K}],$$
$$\overline{\langle M\rangle\mathtt{inr}} = \langle\overline{M}\rangle\mathtt{inr}, \quad \overline{\mathtt{snd}[K]} = \mathtt{snd}[\overline{K}],$$
$$\overline{[K]\mathtt{not}} = [\overline{K}]\mathtt{not}, \quad \overline{\mathtt{not}\langle M\rangle} = \mathtt{not}\langle\overline{M}\rangle, \quad \overline{M \bullet K} = \overline{M} \bullet \overline{K},$$
$$\overline{\mathrm{itr}^A_\alpha[K,L]} = \mathtt{a}[\,(\,\lambda(x,\alpha).(x \bullet \overline{K})\,)@\overline{L}\,],$$
$$\overline{\mathtt{in}^{\mu X.A}\langle M\rangle} = \langle\,\lambda(y,\beta).(y \bullet (\,(\mathcal{Q}_Y[X.A] \bullet \mathcal{R}_M\{y,\gamma\}).\gamma@\beta\,))\,\rangle\mathtt{a},$$
where $\mathcal{Q}_Y[X.A]$ is defined as $\lambda y.\lambda(z,\beta).(\,z \bullet \overline{\mathtt{map}^{X.A}_{\mu X.A,Y,\alpha}\{\,x.(y \bullet (x@\alpha)),\beta\,\}}\,)$,
and $\mathcal{R}_M\{N,K\}$ is defined as $(\,\lambda(x,\alpha).(x \bullet \mathtt{a}[N@\alpha])\,)@(\overline{M}@K)$,
$$\overline{\mathrm{coitr}^A_x\langle M,N\rangle} = (\overline{\mathrm{itr}^{(A)^\circ}_x}[(M)^\circ,(N)^\circ])^\circ, \quad \overline{\mathtt{out}^{\nu X.A}[K]} = \overline{(\mathtt{in}^{\mu X.(A)^\circ}\langle(K)^\circ\rangle)}^\circ.$$

We also define the translation of judgments. The context $\overline{\Gamma}$ is defined as $x_1 : \overline{A_1}, \ldots, x_n : \overline{A_n}$ if Γ is $x_1 : A_1, \ldots, x_n : A_n$. The cocontext $\overline{\Delta}$ is defined as $\alpha_1 : \overline{B_1}, \ldots, \alpha_m : \overline{B_m}$ if Δ is $\alpha_1 : B_1, \ldots, \alpha_m : B_m$. The judgment $\overline{\Gamma \vdash \Delta \mid M : A}$ is defined

as $\overline{\Gamma} \vdash \overline{\Delta} \mid \overline{M} : \overline{A}$. The judgment $\overline{K : A \mid \Gamma \vdash \Delta}$ is defined as $\overline{K} : \overline{A} \mid \overline{\Gamma} \vdash \overline{\Delta}$. The judgment $\overline{\Gamma \mid S \vdash \Delta}$ is defined as $\overline{\Gamma} \mid \overline{S} \vdash \overline{\Delta}$.

The next lemma shows that this translation commutes with $(-)^\circ$.

Lemma 2. *$(\overline{A})^\circ = \overline{(A)^\circ}$, $(\overline{D})^\circ = \overline{(D)^\circ}$, and $(\overline{J})^\circ = \overline{(J)^\circ}$ hold.*

The claim for D is proved by induction on $\mathtt{deg}(D)$.

The next proposition says the translation $\overline{(-)}$ preserves provability.

Proposition 8. *If J is provable in $\mathtt{DC}\mu\nu$, then $\overline{J}$ is provable in $\mathtt{DC2}$.*

This is shown by induction on the degree of the principal expression in J.

Lemma 3. *$\overline{A[B/X]} = \overline{A}[\overline{B}/X]$, $\overline{D}[\overline{M}/x] = \overline{D[M/x]}$, and $\overline{D}[\overline{K}/\alpha] = \overline{D[K/\alpha]}$ hold.*

The first claim is shown by induction on A. The second and the third claims are shown by induction on $\mathtt{deg}(D)$.

The translation $\overline{(-)}$ maps one-step reduction to one or more steps of reduction.

Proposition 9. *For expressions D and E of $\mathtt{DC}\mu\nu$, $D \longrightarrow_{\mathtt{DC}\mu\nu} E$ implies $\overline{D} \longrightarrow^+_{\mathtt{DC2}} \overline{E}$.*

Proof. First we show the claim without $(\beta\mu)$ nor $(\beta\nu)$ by induction on $\longrightarrow_{\mathtt{DC}\mu\nu}$ with Lemma 3. Next, by using this and Lemma 3, we show the claim of this proposition by induction on $\longrightarrow_{\mathtt{DC}\mu\nu}$. We consider cases according to the reduction rule.

Case $(\beta\mu)$. We have $\overline{\mathtt{in}^{\mu X.A}\langle M\rangle \bullet \mathrm{itr}^B_\alpha[K, L]} = \overline{\mathtt{in}^{\mu X.A}\langle M\rangle} \bullet \overline{\mathrm{itr}^B_\alpha[K, L]} = \langle \lambda(y, \beta).(y \bullet ((Q_Y[X.A] \bullet \mathcal{R}_M\{y, \gamma\}).\gamma@\beta))\rangle \mathtt{a} \bullet \mathtt{a}[(\lambda(x.\alpha).(x \bullet \overline{K}))@\overline{L}] \longrightarrow_{\mathtt{DC2}} (\lambda(y, \beta).(y \bullet ((Q_Y[X.A] \bullet \mathcal{R}_M\{y, \gamma\}).\gamma@\beta))) \bullet ((\lambda(x.\alpha).(x \bullet \overline{K}))@\overline{L}) \longrightarrow^+_{\mathtt{DC2}} (\lambda(x.\alpha).(x \bullet \overline{K})) \bullet ((Q_Y[X.A] \bullet \mathcal{R}_M\{\lambda(x.\alpha).(x \bullet \overline{K}), \gamma\}).\gamma@\overline{L}) \longrightarrow^+_{\mathtt{DC2}} (Q_Y[X.A] \bullet \mathcal{R}_M\{\lambda(x.\alpha).(x \bullet \overline{K}), \gamma\}).\gamma \bullet \overline{K}[\overline{L}/\alpha] \longrightarrow_{\mathtt{DC2}} Q_Y[X.A] \bullet \mathcal{R}_M\{\lambda(x.\alpha).(x \bullet \overline{K}), \overline{K}[\overline{L}/\alpha]\} = Q_Y[X.A] \bullet \mathcal{R}_M\{\lambda(x.\alpha).(x \bullet \overline{K}), \overline{K[L/\alpha]}\}$ by Lemma 3. The last expression equals $Q_Y[X.A] \bullet ((\lambda(x_1, \alpha_1).(x_1 \bullet \mathtt{a}[\lambda(x.\alpha).(x \bullet \overline{K})@\alpha_1]))@(\overline{M}@\overline{K[L/\alpha]})) = Q_Y[X.A] \bullet ((\lambda(x_1, \alpha_1).(x_1 \bullet \mathrm{itr}^B_\alpha[K, \alpha_1]))@(\overline{M}@\overline{K[L/\alpha]})) = (\lambda y.\lambda(z, \gamma).(z \bullet \overline{\mathtt{map}^{X.A}_{\mu X.A, Y\beta}\{x.(y \bullet (x@\beta)), \gamma\}})) \bullet ((\lambda(x_1, \alpha_1).(x_1 \bullet \mathrm{itr}^B_\alpha[K, \alpha_1]))@(\overline{M}@\overline{K[L/\alpha]})) \longrightarrow_{\mathtt{DC2}} (\lambda(z, \gamma).(z \bullet \overline{\mathtt{map}^{X.A}_{\mu X.A, Y\beta}\{x.(y \bullet (x@\beta)), \gamma\}[\lambda(x_1, \alpha_1).(x_1 \bullet \mathrm{itr}^B_\alpha[K, \alpha_1])/y]})) \bullet (\overline{M}@\overline{K[L/\alpha]}) = (\lambda(z, \gamma).(z \bullet \overline{\mathtt{map}^{X.A}_{\mu X.A, Y\beta}\{x.((\lambda(x_1, \alpha_1).(x_1 \bullet \mathrm{itr}^B_\alpha[K, \alpha_1])) \bullet (x@\beta)), \gamma\}})) \bullet (\overline{M}@\overline{K[L/\alpha]})$ by Lemma 3. It reduces by $\longrightarrow^+_{\mathtt{DC2}}$ to $(\lambda(z, \gamma).(z \bullet \overline{\mathtt{map}^{X.A}_{\mu X.A, Y\beta}\{\mathrm{itr}^B_\alpha[K, \beta], \gamma\}})) \bullet (\overline{M}@\overline{K[L/\alpha]})$ by the first claim. It reduces by $\longrightarrow^+_{\mathtt{DC2}}$ to $\overline{M \bullet \mathtt{map}^{X.A}_{\mu X.A, Y\beta}\{\mathrm{itr}^B_\alpha[K, \beta], K[L/\alpha]\}}$ by Lemma 3.

Case $(\beta\nu)$ is similar to Case $(\beta\mu)$. Other cases are shown straightforwardly. □

Finally, we complete a proof of strong normalization of $\mathtt{DC}\mu\nu$.

Theorem 4 (Strong normalization of $\mathtt{DC}\mu\nu$). *Every typable expression of $\mathtt{DC}\mu\nu$ is strongly normalizing.*

Proof. Assume that D is typable in $\mathtt{DC}\mu\nu$ and there is an infinite reduction sequence $D \longrightarrow_{\mathtt{DC}\mu\nu} D_1 \longrightarrow_{\mathtt{DC}\mu\nu} \ldots$ starting from D. Then $\overline{D}$ is typable in $\mathtt{DC2}$ by Proposition 8 and $\overline{D} \longrightarrow^+_{\mathtt{DC2}} \overline{D_1} \longrightarrow^+_{\mathtt{DC2}} \ldots$ is an infinite reduction sequence starting from $\overline{D}$ by Proposition 9. This contradicts Theorem 3.

References

1. Barbanera, F., Berardi, S.: A symmetric lambda calculus for classical program extraction. Information and Computation 125(2), 103–117 (1996)
2. Buchholz, W., Feferman, S., Pohlers, W., Sieg, W.: Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoretical Studies. Lecture Notes in Mathematics, vol. 897. Springer, Heidelberg (1981)
3. Curien, P.-L., Herbelin, H.: The duality of computation. In: Proceedings of the 5th ACM SIGPLAN International Conference on Functional Programming, ICFP, pp. 233–243. ACM, New York (2000)
4. Geuvers, H.: Inductive and Coinductive Types with Iteration and Recursion. In: Proceedings of the 1992 workshop on Types for Proofs and Programs, TYPES, pp. 183–207 (1992)
5. Kakutani, Y.: Duality between Call-by-Name Recursion and Call-by-Value Iteration. In: Bradfield, J.C. (ed.) CSL 2002 and EACSL 2002. LNCS, vol. 2471, pp. 506–521. Springer, Heidelberg (2002)
6. Kimura, D.: Call-by-Value is Dual to Call-by-Name, Extended. In: Shao, Z. (ed.) APLAS 2007. LNCS, vol. 4807, pp. 415–430. Springer, Heidelberg (2007)
7. Kimura, D.: Duality between Call-by-value Reductions and Call-by-name Reductions. IPSJ Journal 48(4), 1721–1757 (2007)
8. Mendler, N.P.: Inductive Types and Type Constraints in the Second-Order lambda Calculus. Annals of Pure and Applied Logic 51(1-2), 159–172 (1991)
9. Momigliano, A., Tiu, A.: Induction and co-induction in sequent calculus. In: Berardi, S., Coppo, M., Damiani, F. (eds.) TYPES 2003. LNCS, vol. 3085, pp. 293–308. Springer, Heidelberg (2004)
10. Nordström, B., Petersson, K., Smith, J.M.: Programming in Martin-Löf's Type Theory. Oxford University Press, Oxford (1990)
11. Parigot, M.: Strong normalization for second order classical natural deduction. Journal of Symbolic Logic 62(4), 1461–1479 (1997)
12. Parigot, M.: Strong normalization of second order symmetric lambda-calculus. In: Kapoor, S., Prasad, S. (eds.) FST TCS 2000. LNCS, vol. 1974, pp. 442–453. Springer, Heidelberg (2000)
13. Paulin-Mohring, C.: Inductive Definitions in the System Coq — Rules and Properties. In: Bezem, M., Groote, J.F. (eds.) TLCA 1993. LNCS, vol. 664, pp. 328–345. Springer, Heidelberg (1993)
14. Selinger, P.: Control Categories and Duality: on the Categorical Semantics of the Lambda-Mu Calculus. Mathematical Structures in Computer Science, pp. 207–260 (2001)
15. Tatsuta, M.: Realizability interpretation of coinductive definitions and program synthesis with streams. Theoretical Computer Science 122(1–2), 119–136 (1994)
16. Wadler, P.: Call-by-Value is Dual to Call-by-Name. In: Proceedings of International Conference on Functional Programming, ICFP, pp. 189–201 (2003)
17. Wadler, P.: Call-by-Value is Dual to Call-by-Name – Reloaded. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 185–203. Springer, Heidelberg (2005)

Comparing Böhm-Like Trees*

Jeroen Ketema

Faculty EEMCS, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
j.ketema@ewi.utwente.nl

Abstract. Extending the infinitary rewriting definition of Böhm-like trees to infinitary Combinatory Reduction Systems (iCRSs), we show that each Böhm-like tree defined by means of infinitary rewriting can also be defined by means of a direct approximant function. In addition, we show that counterexamples exists to the reverse implication.

1 Introduction

In λ-calculus, a Böhm tree defines a denotational semantics based on syntax. Essentially, a Böhm tree of a term can been seen as an infinite normal form of the term, omitting subterms that do not 'compute' anything.

What constitutes a term that 'computes' something is not universally determined. Within λ-calculus three alternatives exist: the head normal forms [2, 3], the weak head normal forms [4, 5], and the root-stable terms [6]. These define, besides the Böhm trees, the Lévy-Longo trees and the Berarducci trees. As a result, abstract definitions have appeared that are parameterized over the set of terms 'computing' something. These are the so-called Böhm-like trees [1].

The abstract definitions can be divided into two classes, based on the concrete definition taken as a starting point. One is based on infinitary rewriting [7, 8]; the other is based on so-called direct approximants [1, 9, 10].

Infinitary Rewriting. Within this class, a Böhm-like tree *is* a normal form, albeit not in the original (finite) system but in an infinite system. The infinite system extends the finite one with infinite terms and infinite reductions. Rules are added rewriting terms *not* 'computing' anything — the meaningless terms — to a fresh nullary symbol $\bot$. Pivotal are a number of conditions on the set of meaningless terms guaranteeing that each term has a unique normal form.

Direct Approximants. Within this class, terms are partially ordered by adding a fresh nullary function symbol $\bot$. The direct approximant function — the parameterized component — replaces by $\bot$ any subterm that either reduces to a redex or does *not* 'compute' anything. This yields a normal form that approximates the Böhm-like tree. The tree is obtained by gathering the direct approximants of all the reducts of a term and taking the least upper bound. Pivotal are a number of conditions on the direct approximant function guaranteeing that the least upper bound exists.

* This paper extends earlier unpublished work from the author's Ph.D. thesis [1].

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 239–254, 2009.
© Springer-Verlag Berlin Heidelberg 2009

We show that the direct approximant approach is more expressive than the infinitary rewriting one in the context of Combinatory Reduction Systems (CRSs): Each Böhm-like tree defined by means of infinitary rewriting can also be defined by means of a direct approximant function. The reverse, however, does not hold.

Overview. In Sect. 2 we give some preliminaries, mostly regarding infinitary Combinatory Reduction Systems (iCRSs). In Sect. 3 we extend to iCRSs the infinitary rewriting approach to Böhm-like trees. In Sect. 4 we compare, after shortly reviewing direct approximants. Finally, in Sect. 5 we conclude.

2 Preliminaries

We outline some basic facts concerning iCRSs; see [11, 12, 13] for more detailed accounts. Throughout, we denote the first infinite ordinal by ω, and arbitrary ordinals by α, β, γ, etc. By $\mathbb{N}$ we denote the natural numbers including zero.

Terms and Substitutions. Let Σ be a signature with each element of finite arity. Moreover, assume a countably infinite set of variables and, for each finite arity, a countably infinite set of meta-variables — countably infinite sets suffice given 'Hilbert hotel'-style renaming.

Infinite terms are usually defined by metric completion [11]. Here, we give the shorter, but equivalent, definition from [12]:

Definition 2.1. *The set of* meta-terms *is defined by interpreting the following rules coinductively, where s and $s_1, \ldots, s_n$ are again meta-terms:*

1. *each variable x is a meta-term,*
2. *if x is a variable, then $[x]s$ is a meta-term,*
3. *if Z is an n-ary meta-variable, then $Z(s_1, \ldots, s_n)$ is a meta-term, and*
4. *if $f \in \Sigma$ is n-ary, then $f(s_1, \ldots, s_n)$ is a meta-term.*

The set of finite *meta-terms, a subset of the set of meta-terms, is the set inductively defined by the above rules. A* term *is a meta-term without meta-variables. A* context *is a meta-term over $\Sigma \cup \{\Box\}$ and a* partial meta-term *is a meta-term over $\Sigma_\perp = \Sigma \cup \{\perp\}$, with $\Box$ and $\perp$ fresh nullary function symbols.*

We consider (meta-)terms modulo α-equivalence. A meta-term of the form $[x]s$ is called an *abstraction*; a variable x in s is called *bound* in $[x]s$. Meta-terms with meta-variables only occur in rewrite rules; rewriting itself is defined over terms.

Partial meta-terms are partially ordered where $\perp \preccurlyeq s$ for each partial meta-term s and such that term formation is monotonic modulo α-equivalence [7, 1].

The set of *positions* [11] of a meta-term s, denoted $\mathcal{P}os(s)$, is a set of *finite* strings over $\mathbb{N}$, with each string denoting the 'location' of a subterm in s. If p is a position of s, then $s|_p$ is the *subterm of s at* position p. The length of p is denoted by $|p|$. There exists a well-founded order $<$ on positions: $p < q$ iff p is a proper prefix of q. The concatenation of positions p and q is denoted by $p \cdot q$.

A *valuation* [14], denoted $\bar{\sigma}$, substitutes terms for meta-variables in meta-terms and is defined by coinductively interpreting the rules of valuations for CRSs [11]. In CRSs, applying a valuation to a meta-term yields a unique term. This is not the case for iCRSs [11]. To alleviate this problem, the set of meta-terms satisfying the so-called 'finite chains property' is defined in [11]:

Definition 2.2. *Let s be a meta-term. A* chain *in s is a sequence of (context, position)-pairs $(C_i[\Box], p_i)_{i<\alpha}$, with $\alpha \leq \omega$, such that for each $(C_i[\Box], p_i)$ there exists a term t_i with $C_i[t_i] = s|_{p_i}$ and $p_{i+1} = p_i \cdot q$ where q is the position of the hole in $C_i[\Box]$. A* chain of meta-variables *in s is such that for each $i < \alpha$ it holds that $C_i[\Box] = Z(t_1, \ldots, t_n)$ with $t_j = \Box$ for exactly one $1 \leq j \leq n$.*

The meta-term s is said to satisfy the finite chains property *if no infinite chain of meta-variables occurs in s.*

Remark that $\Box$ only occurs in $C_i[\Box]$ if $i + 1 < \alpha$, otherwise $C_i[\Box] = s|_{p_i}$. The meta-term $[x_1]Z_1([x_2]Z_2(\ldots[x_n]Z_n(\ldots)))$ e.g. satisfies the finite chains property, while $Z(Z(\ldots Z(\ldots)))$ does not. Finite meta-terms always satisfy the finite chains property. The following is shown in [11]:

Proposition 2.3. *Let s be a meta-term satisfying the finite chains property and let $\bar{\sigma}$ be a valuation. There is a unique term that is the result of applying $\bar{\sigma}$ to s.*

Rewriting. Recall that a *pattern* is a finite meta-term each meta-variable of which has distinct bound variables as arguments and that a meta-term is *closed* if all variables occur bound [14].

Definition 2.4. *A* rewrite rule *is a pair of closed meta-terms (l, r), denoted $l \to r$, with l a finite pattern of the form $f(s_1, \ldots, s_n)$ and r satisfying the finite chains property such that all meta-variables that occur in r also occur in l.*

An infinitary Combinatory Reduction System (iCRS) *is a pair $\mathcal{C} = (\Sigma, R)$ with Σ a signature and R a set of rewrite rules.*

Left-linearity and orthogonality are defined as for CRSs [14] (left-hand sides of rewrite rules are finite). A rewrite rule is *collapsing* if the root of its right-hand side is a meta-variable. Moreover, a pattern is *fully-extended*, if, for each meta-variable Z and abstraction $[x]s$ with an occurrence of Z in its scope, x is an argument of that occurrence of Z; a rewrite rule is *fully-extended* if its left-hand side is and an iCRS is *fully-extended* if all its rewrite rules are.

Definition 2.5. *A* rewrite step *is a pair of terms (s, t) denoted $s \to t$ and adorned with a context $C[\Box]$, a rewrite rule $l \to r$, and a valuation $\bar{\sigma}$ such that $s = C[\bar{\sigma}(l)]$ and $t = C[\bar{\sigma}(r)]$. The term $\bar{\sigma}(l)$ is called an $l \to r$-*redex *and* occurs *at position p and depth $|p|$ in s, where p is the position of the hole in $C[\Box]$.*

A position q of s occurs in the redex pattern *of the redex at position p if $q \geq p$ and if there does not exist a position q' with $q \geq p \cdot q'$ such that q' is the position of a meta-variable in l.*

Above, $\bar{\sigma}(l)$ and $\bar{\sigma}(r)$ are well-defined, as both left- and right-hand sides of rewrite rules satisfy the finite chains property (left-hand sides as they are finite).

We say that a redex s *overlaps* a term t at position p, if p occurs in the redex pattern of s and $s|_p = t$ [7]. Moreover a redex and a rewrite step are *collapsing* if the employed rewrite rule is. Using rewrite steps, we define reductions:

Definition 2.6. *A* transfinite reduction *with domain $\alpha > 0$ is a sequence of terms $(s_\beta)_{\beta<\alpha}$ such that $s_\beta \to s_{\beta+1}$ for all $\beta + 1 < \alpha$. For each $s_\beta \to s_{\beta+1}$, let d_β denote the depth of the contracted redex. The reduction is* strongly convergent *if α is a successor ordinal and if for every limit ordinal $\gamma \leq \alpha$ it holds that s_β converges to s_γ and d_β tends to infinity in case β approaches γ from below.*

Consider the rules $a \to a$ and $f(Z) \to g(f(Z))$. The reduction

$$f(a) \to g(f(a)) \to \cdots \to g^n(f(a)) \to \cdots g^\omega\,,$$

with g^ω denoting $g(g(\ldots g(\ldots)))$, is strongly convergent. The reduction

$$f(a) \to f(a) \to \cdots \to f(a) \to \cdots$$

is not strongly convergent, as each contracted redex occurs at depth 1.

By $s \twoheadrightarrow^\alpha t$, resp. $s \twoheadrightarrow^{\leq\alpha} t$, we denote a strongly convergent reduction of length α, resp. of length at most α. By $s \twoheadrightarrow t$, resp. $s \to^* t$, we denote a strongly convergent reduction of arbitrary length, resp. of finite length.

Across strongly convergent reductions we assume that a position that occurs in the redex pattern of a contracted redex does not have any descendants; likewise for residuals [11]. We write $P/(s \twoheadrightarrow t)$ for the descendants of a set of positions $P \subseteq \mathcal{P}os(s)$ across a strongly convergent reduction $s \twoheadrightarrow t$ and $\mathcal{U}/(s \twoheadrightarrow t)$ for the residuals of a set $\mathcal{U}$ of subterms of s across $s \twoheadrightarrow t$.

Below, we appeal to a number of properties of iCRSs. The first is compression:

Theorem 2.7 (Compression [11]). *For every fully-extended, left-linear iCRS, if $s \twoheadrightarrow^\alpha t$, then $s \twoheadrightarrow^{\leq\omega} t$.*

A term s is *hypercollapsing*, resp. *root-active*, if for all $s \twoheadrightarrow t$ there exists a $t \twoheadrightarrow t'$ such that t' is a collapsing redex, resp. a redex. We write $s \sim_{hc} t$ if t can be obtained from s by replacing hypercollapsing subterms in s by other hypercollapsing subterms.

Let $\sim$ be an equivalence relation. Confluence modulo $\sim$ means that if $s \twoheadrightarrow s'$ and $t \twoheadrightarrow t'$ with $s \sim t$, then $s' \twoheadrightarrow s''$ and $t' \twoheadrightarrow t''$ with $s'' \sim t''$. For $\sim_{hc}$ we have:

Theorem 2.8. *Given a fully-extended, orthogonal iCRSs, the relation $\sim_{hc}$ is an equivalence relation and the system is confluent modulo $\sim_{hc}$.*

The above is shown in [12] under assumption that rewrite rules have finite right-hand sides; in [13] the result is extended to allow for infinite right-hand sides.

3 Infinitary Rewriting

We extend the infinitary rewriting approach to Böhm-like trees from [7, 8] to fully-extended, orthogonal iCRSs. Given an iCRS and a set of so-called meaningless terms, this means we define a confluent and normalising rewrite system.

Following the pattern laid down in [7, 8], we start in Sect. 3.1 by stating a number of axioms for sets of meaningless terms. Assuming some of the axioms, we consider 'meaningful' terms in Sect. 3.2 and we define Böhm-like trees in Sect. 3.3. In Sect. 3.4, we construct a set of partial terms given a set of terms and show the axioms are preserved. Finally, in Sect. 3.5, we consider some examples, some of which employ the construction from the Sect. 3.4.

3.1 Axioms

To state our axioms, assume $\mathcal{U}$ is a set of terms. We call the terms in this set *meaningless*; intuitively they are not supposed to 'compute' anything.

Let s and t be terms with $P \subseteq \mathcal{P}os(s)$ such that $s|_p \in \mathcal{U}$ for each $p \in P$. We write $s \to_P^{\mathcal{U}} t$, resp. $s \leftrightarrow_P^{\mathcal{U}} t$, if t can be obtained from s by replacing the subterms at positions in P by arbitrary terms, resp. by terms from $\mathcal{U}$. Remark that $\leftrightarrow^{\mathcal{U}}$ is reflexive and symmetric, i.e. $s \leftrightarrow_P^{\mathcal{U}} s$ and $s \leftrightarrow_P^{\mathcal{U}} t$ iff $t \leftrightarrow_P^{\mathcal{U}} s$. We write $s \to^{\mathcal{U}} t$ and $s \leftrightarrow^{\mathcal{U}} t$ if the set of positions is irrelevant or clear from the context.

The considered axioms stem from [8] and are as follows:

Residuals If $s \twoheadrightarrow t$ and $s|_p \in \mathcal{U}$, then $t|_q \in \mathcal{U}$ for all $q \in p/(s \twoheadrightarrow t)$.
Overlap If a redex s overlaps a term in $\mathcal{U}$, then $s \in \mathcal{U}$.
Root-activeness If s is root-active, then $s \in \mathcal{U}$.
Hypercollapsingness If s is hypercollapsing, then $s \in \mathcal{U}$.
Indiscernability If $s \leftrightarrow^{\mathcal{U}} t$, then $s \in \mathcal{U}$ iff $t \in \mathcal{U}$.

Intuitively, residuals and overlap state, resp., that no information can be obtained about meaningless terms by reducing them or by placing them in a context. All root-active terms, which includes all hypercollapsing terms, reduce indefinitely at the root and do not become stable. Hence, it is reasonable to consider these terms to be meaningless. This will also guarantee the existence of normal forms later on. Indiscernability states that the identities of the meaningless subterms of a meaningless term are irrelevant.

Indiscernability coincides with transitivity, as shown in [8, Lemma 12.9.17]:

Lemma 3.1. *A set $\mathcal{U}$ satisfies indiscernability iff $\leftrightarrow^{\mathcal{U}}$ is transitive.*

Hence, in case $\mathcal{U}$ satisfies indiscernability, $\leftrightarrow^{\mathcal{U}}$ is an equivalence relation.

The next lemma introduces two derived axioms describing the simulation of one reduction by another. These axioms are used extensively in the remainder.

Lemma 3.2. *In a fully-extended, left-linear iCRS, if $\mathcal{U}$ satisfies residuals and overlap, then for $s \twoheadrightarrow s'$:*

Simulation *if $s \to^{\mathcal{U}} t$, there exists a term t' such that $t \twoheadrightarrow t'$ and $s' \to^{\mathcal{U}} t'$, and*
Bisimulation *if $s \leftrightarrow^{\mathcal{U}} t$, there exists a term t' such that $t \twoheadrightarrow t'$ and $s' \leftrightarrow^{\mathcal{U}} t'$.*

Proof (Sketch). By ordinal induction on the length of $s \twoheadrightarrow s'$, using fully-extendedness and left-linearity. Employ the fact that each subterm in $\mathcal{U}$ has a residual — a redex pattern either occurs fully inside or fully outside a subterm in $\mathcal{U}$ by overlap — and the fact that each residual of a subterm in $\mathcal{U}$ is in $\mathcal{U}$ — by residuals. □

3.2 Meaningful Terms

Meaningless subterms can occur in the reducts of a term s — even without s having meaningless subterms itself. In such a case, s cannot be called completely meaningful. Contrary, any term not possessing this property can be considered meaningful. Following [7] and assuming an iCRS $\mathcal{C}$ and set of terms $\mathcal{U}$, we define:

Definition 3.3. *A term s is* totally meaningful *if no subterm of any reduct of s occurs in $\mathcal{U}$.*

With the help of totally meaningful terms, we can express the intuition that meaningless terms should be "computational irrelevant" [7]:

Definition 3.4. *The set $\mathcal{U}$ is called* generic*, if for every $s \in \mathcal{U}$ and context $C[\Box]$ reduction of $C[s]$ to a totally meaningful term implies reduction of $C[t]$ to a totally meaningful term for every term t.*

Simulation is a sufficient criterion for genericity to hold:

Theorem 3.5 (Genericity). *If $\mathcal{C}$ is fully-extended and left-linear and $\mathcal{U}$ satisfies simulation, then $\mathcal{U}$ is generic.*

Proof. Let $C[s] \twoheadrightarrow s'$ with $s \in \mathcal{U}$ and s' totally meaningful. If t is arbitrary, then $C[s] \to^{\mathcal{U}} C[t]$. Hence, by simulation a term t' exists such that $s' \to^{\mathcal{U}} t'$. Since s' is totally meaningful, $s' = t'$ and genericity follows. □

Above, "computational irrelevancy" is expressed employing reduction. Alternatively, it can be expressed employing conversion; in which case we define [7]:

Definition 3.6. *The iCRS $\mathcal{C}$ is* relative consistent *given $\mathcal{U}$, if $s\ (\twoheadrightarrow \cdot \leftrightarrow^{\mathcal{U}} \cdot \twoheadleftarrow)^*\ t$ implies $s\ (\twoheadrightarrow \cdot \twoheadleftarrow)^*\ t$ for all totally meaningful terms s and t.*

To show that relative consistency holds under the assumption of certain axioms, we first state a confluence theorem:

Theorem 3.7 (Confluence). *If $\mathcal{C}$ is fully-extended and orthogonal and $\mathcal{U}$ satisfies bisimulation, hypercollapsingness, and indiscernability, then $\mathcal{C}$ is confluent modulo $\mathcal{U}$.*

The proof is similar to that of [7, Lemma 23], observing Lemma 3.1 and using bisimulation instead of [7, Lemma 21]. Lemma 14 in [7] is Theorem 2.8.

We can now show relative consistency. Remark that the assumed axioms are much stronger than in the case of genericity.

Theorem 3.8 (Relative Consistency). *If $\mathcal{C}$ is fully-extended and orthogonal and $\mathcal{U}$ satisfies bisimulation, hypercollapsingness, and indiscernability, then $\mathcal{C}$ is relatively consistent given $\mathcal{U}$.*

Proof. Let $s\ (\twoheadrightarrow \cdot \leftrightarrow^{\mathcal{U}} \cdot \twoheadleftarrow)^*\ t$, with s and t totally meaningful. By induction on the number of changes in the direction of the rewrite relation in $s\ (\twoheadrightarrow \cdot \leftrightarrow^{\mathcal{U}} \cdot \twoheadleftarrow)^*\ t$ and Theorem 3.7 there exist terms s' and t' such that $s \twoheadrightarrow s' \leftrightarrow^{\mathcal{U}} t' \twoheadleftarrow t$. Hence, since s and t are totally meaningful, $s' = t'$ and the result follows. □

3.3 Böhm-Like Trees

In this section, we define Böhm-like trees by means of infinitary rewriting. The definition proceeds in two steps. In the first, we define an iCRS that extends the iCRS whose Böhm-like trees we want to define. In the second step, we give sufficient criteria — in the form of our axioms — implying that the defined iCRS is confluent and normalising. Confluence and normalisation imply that each term has a unique normal form, the Böhm-like tree of that term.

We assume that our set of meaningless terms is a set of *partial* terms, we denote this set by $\mathcal{U}_\bot$. In the next section, we show how to obtain such a set of partial terms from a set of (non-partial) terms.

Our iCRS and Böhm-like tree are defined as follows:

Definition 3.9. *The* Böhm-like iCRS *of an iCRS* $\mathcal{C} = (\Sigma, R)$ *and a set of partial terms* $\mathcal{U}_\bot$ *is a pair* $\mathcal{B} = (\Sigma_\bot, R \cup B)$ *with* $B = \{b \to_\bot \bot \mid b \in \mathcal{U}_\bot,\ b \neq \bot\}$.

A rewrite step *in* $\mathcal{B}$ *is a pair of partial terms* (s, t) *denoted* $s \to t$ *and adorned with a context* $C[\Box]$ *and a rule* $l \to r \in R$ *or a rule* $b \to_\bot \bot \in B$ *such that:*

- $s = C[\bar\sigma(l)]$ *and* $t = C[\bar\sigma(r)]$ *with* $\bar\sigma$ *a valuation, or*
- $s = C[b]$ *and* $t = C[\bot]$.

A Böhm-like tree *of a partial term* s *is a normal form of* s *with respect to* $\mathcal{B}$.

Remark that the definition of rewrite steps deviates slightly from the usual one; no valuation is employed in case the rule originates from B. Reduction-wise nothing changes; we employ strongly convergent reductions.

Writing $s \twoheadrightarrow_R t$ for a reduction in case all rewrite rules originate from the set R, we have the following:

Lemma 3.10. *Given a fully-extended, left-linear iCRS and a set* $\mathcal{U}_\bot$*:*

1. *if* $\mathcal{U}_\bot$ *satisfies root-activeness, then every term has a Böhm-like tree, and*
2. *if* $\mathcal{U}_\bot$ *satisfies residuals, then* $s \twoheadrightarrow t$ *implies* $s \twoheadrightarrow_R \cdot \twoheadrightarrow_\bot t$.

The proof of the first part, resp. of the second part, is identical to that of [7, Theorem 1], resp. [7, Lemma 27].

The following now suffices to ensure that each (partial) term has a unique Böhm-like tree.

Theorem 3.11. *Given a fully-extended, orthogonal iCRS, if* $\mathcal{U}_\bot$ *satisfies residuals, overlap, root-activeness, and indiscernability and if* $\bot \in \mathcal{U}_\bot$*, then* $\mathcal{B}$ *is confluent and every term has a unique Böhm-like tree.*

The proof is similar to that of [7, Theorem 2], using Lemma 3.10 instead of Theorem 1 and Lemma 27 in [7], Theorem 2.8 instead of Lemma 14 in [7], and Lemma 3.1 instead of Lemma 15 in [7].

In case the Böhm-like tree of a term s is uniquely defined by the set $\mathcal{U}_\bot$, we denote it by $\mathrm{BLT}^\infty(s)$. The following is immediate by the previous theorem:

Corollary 3.12 (Congruence). *Given a fully-extended, orthogonal iCRS, if* $\mathcal{U}_\perp$ *satisfies residuals, overlap, root-activeness, and indiscernability and if* $\perp \in \mathcal{U}_\perp$*, then for all terms* s *and* t *and each context* $C[\Box]$ *it holds that* $\mathrm{BLT}^\infty(s) = \mathrm{BLT}^\infty(t)$ *implies* $\mathrm{BLT}^\infty(C[s]) = \mathrm{BLT}^\infty(C[t])$.

Remark 3.13. Overlap can be replaced by bisimulation in the above theorem. Doing so, we can prove uniqueness of Böhm-like trees for certain iCRSs and sets $\mathcal{U}_\perp$ where overlap *does* occur.

Consider for example the rule:

$$f(g(Z)) \to f(Z)$$

and the set

$$\mathcal{U}_\perp = \{g^n(\perp),\ f(g^\omega) \mid n \in \mathbb{N}\}\,.$$

Residuals, root-activeness, and indiscernability follow easily. Concerning bisimulation, the only interesting case is $f(g^{n+1}(\perp)) \to f(g^n(\perp))$ with $f(g^{n+1}(\perp)) \leftrightarrow^{\mathcal{U}} f(g^m(\perp))$. As $g^n(\perp) \in \mathcal{U}$ for all $n \in \mathbb{N}$, we have the following diagram:

$$\begin{array}{ccc} f(g^{n+1}(\perp)) & \overset{\mathcal{U}}{\longleftrightarrow} & f(g^m(\perp)) \\ \downarrow & & \| \\ f(g^n(\perp)) & \overset{\mathcal{U}}{\longleftrightarrow} & f(g^m(\perp)) \end{array}$$

Thus, bisimulation holds. As $0 \in \mathbb{N}$, we also have $\perp \in \mathcal{U}_\perp$. Hence, we find that every term has a unique Böhm-like tree although $\mathcal{U}_\perp$ does not satisfy overlap.

3.4 Extending $\mathcal{U}$ with $\perp$

Assume we have at our disposal an iCRS $\mathcal{C} = (\Sigma, R)$ and a set of (non-partial) terms $\mathcal{U}$. We next define a set of partial terms $\mathcal{U}_\perp \supseteq \mathcal{U}$ [7]. The set is defined in such a way that each of the axioms satisfied by $\mathcal{U}$ is also satisfied by $\mathcal{U}_\perp$. The construction slightly simplifies some of our examples in the next section.

Definition 3.14. *A* $\perp$-instance *of a partial term* s *is a term* t *obtained by replacing every* $\perp$ *in* s *by a term in* $\mathcal{U}$*, i.e.* $s \leftarrow_P^{\mathcal{U}} t$*, where* $P = \{p \in \mathcal{P}os(s) \mid s|_p = \perp\}$.

The set $\mathcal{U}_\perp$ *is the union of* $\{\perp\}$ *and the set of partial terms each of which has a* $\perp$*-instance in* $\mathcal{U}$.

Note that if t is a $\perp$-instance of s, then $s \preccurlyeq t$; the reverse does not necessarily hold. Explicit inclusion of $\perp$ in $\mathcal{U}_\perp$ only makes difference in case $\mathcal{U}$ is empty, we then have $\mathcal{U}_\perp = \{\perp\}$. Otherwise, $\perp$ is included automatically as each term in $\mathcal{U}$ is a $\perp$-instance of $\perp$. Inclusion of $\{\perp\}$ is needed in light of Theorem 3.11.

As promised, we have the following:

Lemma 3.15. *For each of residuals, overlap, root-activeness, hypercollapsingness, and indiscernability, if* $\mathcal{U}$ *satisfies the property, then so does* $\mathcal{U}_\perp$.

Each property in the lemma follows easily; see [7, Lemma 25]. Roughly, we are required to show that each considered partial term has a $\perp$-instance in $\mathcal{U}$.

3.5 Examples

We consider three interesting sets of meaningless terms from [7] defining Böhm-like trees. We show that in the higher-order case these sets also define Böhm-like trees. The sets are those of the root-active, opaque, and Huet-Lévy undefined terms.

Root-Active. As argued above, root-active terms are essentially meaningless. Hence, it is interesting to consider the set solely consisting of these terms. This set defines a Böhm-like tree, given a fully-extended, orthogonal iCRS.

Recall from [15] that a term is root-active iff a perpetual reduction starts from it, i.e. a reduction with an infinite number of root-steps. We have:

Proposition 3.16. *Let s and t be terms. If s is root-active and $s \leftrightarrow^{\mathcal{U}} t$, then t is root-active.*

Proof (Sketch). Since a term is root-active iff it has a perpetual reduction starting from it, consider a perpetual reduction S starting from s and define a perpetual reduction starting from t. To do so, omit those steps from S that occur inside subterms that are residuals of subterms replaced in $s \leftrightarrow^{\mathcal{U}} t$. □

Employing the above, we have the following:

Proposition 3.17. *The root-active terms satisfy residuals, overlap, root-activeness, and indiscernability.*

Proof. Residuals and overlap follow by orthogonality. Root-activeness is immediate by definition. Indiscernability follows by Proposition 3.16. □

The root-active terms also satisfy hypercollapsingness, as every hypercollapsing term is root-active. Simulation and bisimulation follow by Lemma 3.2. Hence, genericity and relative consistency also follow. By Lemma 3.15, each term has a Böhm-like tree with respect to the set each partial terms each of which has a $\bot$-instance that is root-active.

Opaque. Similar to root-activeness, opaqueness takes an axiom as its starting point, in this case overlap. We again assume a fully-extended, orthogonal iCRS.

Definition 3.18. *A closed term s is* opaque *iff no term to which s reduces is overlapped by a redex at a non-root position. A term is* opaque *iff every closed substitution instance is.*

The above definition stems from [7]. The definition in [16], i.e. that a term s is opaque iff no term reachable from s is overlapped by a redex at a non-root position, cannot be the intended one, as it is not closed under substitution in case of λ-calculus. For example, the open term x would then be opaque, while the substitution instance $\lambda y.y$ is not, as it is a subterm of $(\lambda y.y)z$.

We have the following; the proof is identical to the one in Sect. 8.1.3 of [7]:

Proposition 3.19. *The set of opaque terms satisfies residuals, overlap, root-activeness, and indiscernability.*

The opaque terms also satisfy hypercollapsingness, as every hypercollapsing term is root-active. Simulation and bisimulation follow by Lemma 3.2. Hence, genericity and relative consistency also follow. By Lemma 3.15, each term has a Böhm-like tree with respect to the set each partial terms each of which has a $\bot$-instance that is opaque.

Huet-Lévy Undefined. As shown in [7, Sect. 8.1.4], the Huet-Lévy TRS — a starting point for the direct approximant approach [10, 1] — can be used to define a set of meaningless terms. This approach extends to CRSs, assuming a fully-extended, orthogonal CRS $\mathcal{C}$, i.e. only allowing finite terms and reductions.

Definition 3.20. *The* Huet-Lévy CRS *of* $\mathcal{C}$ *is defined as* $\mathcal{HL} = (\Sigma_\bot, HL)$*, with:*

$$HL = \{d \to \bot \mid d \text{ a partial pattern}\} \cup \{l \to \bot \mid l \to r \in R\}\,,$$

where a partial pattern d *is any pattern* $\bot \neq d \preccurlyeq l$ *with* $l \to r \in R$ *such that no valuation* $\bar\sigma$ *exists with* $\bar\sigma(d) = \bar\sigma(l)$*.*

By the definition, Huet-Lévy CRSs are orthogonal, because $\mathcal{C}$ is orthogonal, and without collapsing rules. We easily obtain the following:

Proposition 3.21. *The Huet-Lévy CRS* $\mathcal{HL}$ *of* $\mathcal{C}$ *is confluent. Any finite partial term* s *has a unique normal form* $\omega_{\mathrm{HL}}(s)$ *and for all finite partial terms* s *and* t*:*

1. $\omega_{\mathrm{HL}}(s) \preccurlyeq s$*,*
2. *if a redex occurs at position* p *in* s*, then* $\omega_{\mathrm{HL}}(s) \preccurlyeq s[\bot]_p$*, and*
3. *if* $s \to t$*, then* $\omega_{\mathrm{HL}}(s) \preccurlyeq \omega_{\mathrm{HL}}(t)$*,*

We can now define the following two sets:

$$\mathcal{U}^{\mathrm{f}}_{\mathrm{HL}} = \{s \text{ a finite partial term} \mid \forall s \to^* t : \omega_{\mathrm{HL}}(t) = \bot\}$$
$$\mathcal{U}_{\mathrm{HL}} = \{s \mid \forall s \twoheadrightarrow t \text{ and } u \preccurlyeq t : u \in \mathcal{U}^{\mathrm{f}}_{\mathrm{HL}}\}$$

Proposition 3.22. *The terms in* $\mathcal{U}_{\mathrm{HL}}$ *satisfies residuals, overlap, root-activeness, and indiscernability.*

Proof. Overlap, resp. residuals, follows by orthogonality of the CRS, resp. of the Huet-Lévy CRS. Root-activeness follows by Proposition 3.21(2).

In the case of indiscernability, consider $s \leftrightarrow^{\mathcal{U}_{\mathrm{HL}}} t$ with $s \in \mathcal{U}_{\mathrm{HL}}$ and let $t \twoheadrightarrow t'$. By bisimulation, which follows from Lemma 3.2, there exists a reduction $s \twoheadrightarrow s'$ such that $s' \leftrightarrow^{\mathcal{U}_{\mathrm{HL}}} t'$. Consider any finite partial term $u_t \preccurlyeq t'$. As $s' \leftrightarrow^{\mathcal{U}_{\mathrm{HL}}} t'$, we have a finite partial term $u_s \preccurlyeq s'$ such that $u_s \leftrightarrow^{\mathcal{U}^{\mathrm{f}}_{\mathrm{HL}}} u_t$. Since u_s and u_t are finite and $u_s \leftrightarrow^{\mathcal{U}^{\mathrm{f}}_{\mathrm{HL}}} u_t$, there exists a finite partial term u such that $u_s \to^* u \,{}^*\!\leftarrow u_t$, employing the reduction rules of the Huet-Lévy CRS. Hence, as $s \in \mathcal{U}_{\mathrm{HL}}$ implies $u_s \to^* \bot$, we have $u_t \to^* \bot$ and indiscernability follows. □

The set $\mathcal{U}_{\mathrm{HL}}$ also satisfies hypercollapsingness, as every hypercollapsing term is root-active. Simulation and bisimulation follow by Lemma 3.2. Hence, genericity and relative consistency also follow. Moreover, as $\bot \in \mathcal{U}_{\mathrm{HL}}$, each term has a Böhm-like tree with respect to $\mathcal{U}_{\mathrm{HL}}$.

Remark 3.23. The definition of the set $\mathcal{U}_{\mathrm{HL}}$ differs from the one in [7], which requires an additional nullary function symbol. It is easily shown that $\mathcal{U}_{\mathrm{HL}}$ and the set defined in [7] yield exactly the same Böhm-like tree for each term.

Remark 3.24. Consider the CRS encoding of the β-rule from λ-calculus:

$$\mathtt{app}(\mathtt{lam}([x]Z(x)), Z') \to Z(Z')$$

This rule yields the following Huet-Lévy CRS:

$$\begin{aligned} \mathtt{app}(\mathtt{lam}([x]Z(x)), Z') &\to \bot \\ \mathtt{app}(\mathtt{lam}(\bot), Z') &\to \bot \\ \mathtt{app}(\bot, Z') &\to \bot \end{aligned}$$

Any term that is the encoding of a term from λ-calculus is in $\mathcal{U}_{\mathrm{HL}}$ iff the term does not have a weak head normal form, i.e. the λ-term does not reduce to a term of the form $\lambda x.s$ or $xs_1s_2 \ldots s_n$. Hence, $\mathcal{U}_{\mathrm{HL}}$ defines the Lévy-Longo tree [4,5,17].

This means that not only the opaque terms define an iTRS analogue of Lévy-Longo trees, as stated in [16], but so does $\mathcal{U}_{\mathrm{HL}}$. The set of opaque terms and $\mathcal{U}_{\mathrm{HL}}$ do not need to coincide: Consider a ruleless CRS. All terms are opaque, while $\mathcal{U}_{\mathrm{HL}} = \{\bot\}$. Thus, the question whether an analogue of the Lévy-Longo tree exists for TRSs [10] does not have a unique answer.

4 Comparison

Having defined Böhm-like trees by means of infinitary rewriting, we can now compare this approach with the direct approximant approach. To do so, we first recall the direct approximant definition for CRSs from [9].

Definition 4.1. *Let $\mathcal{C} = (\Sigma, R)$ be an orthogonal CRS. A* direct approximant function *is a map ω on finite partial terms, such that for all terms s and t:*

1. *$\omega(s) \preccurlyeq s$,*
2. *if a redex occurs at position p in s, then $\omega(s) \preccurlyeq s[\bot]_p$, and*
3. *if $s \to t$, then $\omega(s) \preccurlyeq \omega(t)$,*

where $\omega(s)$ is called the direct approximant *of s.*

Hence, ω_{HL}, as defined in Sect. 3.5, is a direct approximant function.

The definition only concerns CRSs and not iCRSs. As such, our comparison only concerns the Böhm-like trees of *finite* terms. Since each pair that defines a CRS also defines an iCRS, with the reductions of the CRS forming a subset of the reductions of the iCRS, this does not pose any obstacle in our comparison.

In the current context, Böhm-like trees are defined as follows:

Definition 4.2. *Let s be a finite partial term. The* Böhm-like tree *of s with respect to ω, denoted* $\mathrm{BLT}(s)$*, is defined as:*

$$\mathrm{BLT}(s) = \bigsqcup \{\omega(t) \mid s \to^* t\}\,.$$

The set $\{\omega(t) \mid s \to^* t\}$ is directed by confluence and the third clause of the direct approximant definition. Hence, the least upper bound exists.

Usually, $\mathrm{BLT}(s)$ is defined by means of downward closure instead of the least upper bound [9, 10, 1], with the (infinite) terms being defined by means of ideal completion. However, downward closure and the least upper bound coincide in case of ideals. Replacing downward closure by the least upper bound allows us to avoid the introduction of (infinite) terms by means of ideal completion, using the isomorphic definition of terms given in Sect. 2 [1].

Obviously, each finite partial term has a unique Böhm-like tree. Moreover, Böhm-like trees are preserved under rewriting:

Theorem 4.3. *If $s \to^* t$, with s and t finite, then* $\mathrm{BLT}(s) = \mathrm{BLT}(t)$.

Proof. Let $s \to^* t$. By confluence of $\mathcal{C}$ there exists for every $s \to^* s'$ and $t \to^* t'$ a partial term u such that $s' \to^* u \,{}^*\!\leftarrow t'$. Hence, by the third clause of Definition 4.1 and the definition of Böhm-like trees we have $\mathrm{BLT}(s) = \mathrm{BLT}(t)$. □

4.1 From Infinitary Rewriting to Direct Approximants

Assume $\mathcal{C} = (\Sigma, R)$ is a fully-extended, orthogonal CRS and $\mathcal{U}$ is a set of meaningless terms satisfying residuals, overlap, root-activeness, and indiscernability such that $\bot \in \mathcal{U}$. We show that we can define a direct approximant function such that for each finite term we have that the Böhm-like tree it defines is identical to the Böhm-like tree we would obtain by means of infinitary rewriting.

We first define a map:

Definition 4.4. *The map $\omega_{\mathcal{U}}$ on finite partial terms is defined for each term s as the largest term t, with respect to the prefix order, such that $t \preccurlyeq s[\bot]_p$ for all $p \in \mathcal{P}os(s)$ with $s|_p$ either transfinitely reducible to a redex or to term in $\mathcal{U}$.*

We now show:

Lemma 4.5. *The map $\omega_{\mathcal{U}}$ defines a direct approximant function.*

Proof. We consider each of the clauses of Definition 4.1 in turn:

1. That $\omega_{\mathcal{U}}(s) \preccurlyeq s$ is immediate by the definition of $\omega_{\mathcal{U}}$.
2. That $\omega_{\mathcal{U}}(s) \preccurlyeq s[\bot]_p$ for all $p \in \mathcal{P}os(s)$ if redex occurs at p in s, follows by the fact that $\omega_{\mathcal{U}}(s) \preccurlyeq s[\bot]_p$ if $s|_p$ transfinitely reduces to a redex.
3. That $s \to t$ implies $\omega_{\mathcal{U}}(s) \preccurlyeq \omega_{\mathcal{U}}(t)$, follows, as for each position p parallel or above the contracted redex (in both s and t), we have that $s|_p$ transfinitely reduces to a redex or to term in $\mathcal{U}$ if $t|_p$ does. □

Write $\mathrm{BLT}^{\infty}_{\mathcal{U}}$ for the Böhm-like tree defined by the Böhm-like iCRS $\mathcal{B}$ of $\mathcal{C}$ and $\mathcal{U}$ and write $\mathrm{BLT}_{\mathcal{U}}$ for the tree defined by $\omega_{\mathcal{U}}$. We show our main result, i.e. coincidence of $\mathrm{BLT}^{\infty}_{\mathcal{U}}$ and $\mathrm{BLT}_{\mathcal{U}}$. The proof effectively defines a bisimulation.

Theorem 4.6. *If s is a finite partial term, then* $\mathrm{BLT}_{\mathcal{U}}(s) = \mathrm{BLT}^{\infty}_{\mathcal{U}}(s)$.

Proof. Given a finite partial term s, we show by induction on positions p that $p \in \mathcal{P}os(\mathrm{BLT}_{\mathcal{U}}(s))$ iff $p \in \mathcal{P}os(\mathrm{BLT}^{\infty}_{\mathcal{U}}(s))$ and $root(\mathrm{BLT}_{\mathcal{U}}(s)|_p) = root(\mathrm{BLT}^{\infty}_{\mathcal{U}}(s)|_p)$.

Obviously, if p is the root position, it is a position of both Böhm-like trees. Moreover, if $p = q \cdot i$, then p is a position of both Böhm-like trees given that q is such a position, with $root(\mathrm{BLT}_{\mathcal{U}}(s)|_q) = root(\mathrm{BLT}^{\infty}_{\mathcal{U}}(s)|_q)$ of arity n and $0 \leq i \leq n$, considering $[x]$ to be a unary function symbol for every variable x. This leaves to show for each position p that $root(\mathrm{BLT}_{\mathcal{U}}(s)|_p) = root(\mathrm{BLT}^{\infty}_{\mathcal{U}}(s)|_p)$.

Suppose $root(\mathrm{BLT}_{\mathcal{U}}(s)|_p) = f$. Either $f = \bot$ or $f \neq \bot$. If $f = \bot$, we have by definition of $\omega_{\mathcal{U}}$ for every $s \to^* t$ with $p \in \mathcal{P}os(\omega_{\mathcal{U}}(t))$ that $t|_p$ transfinitely reduces to a redex or term in $\mathcal{U}$. The first implies $t|_p$ is root-active and, whence, in $\mathcal{U}$. Thus, $root(\mathrm{BLT}^{\infty}_{\mathcal{U}}(s)|_p) = \bot$, as $t|_p$ transfinitely reduces to a term in $\mathcal{U}$ and $p \in \mathcal{P}os(\omega_{\mathcal{U}}(t))$. In case $f \neq \bot$, $s \to^* t$ with $p \in \mathcal{P}os(\omega_{\mathcal{U}}(t))$ and $root(\omega_{\mathcal{U}}(t)|_p) = f$ by definition of $\omega_{\mathcal{U}}$. Hence, again by definition of $\omega_{\mathcal{U}}$, $t|_p$ neither transfinitely reduces to a redex nor to term in $\mathcal{U}$, implying $root(\mathrm{BLT}^{\infty}_{\mathcal{U}}(s)|_p) = f$.

Now suppose $root(\mathrm{BLT}^{\infty}_{\mathcal{U}}(s)|_p) = f$. As before, either $f = \bot$ or $f \neq \bot$. In case $f = \bot$, there exists by Lemma 3.10(2) and compression a reduction $s \to^* t$ such that $p \in \mathcal{P}os(t)$, all $t|_q$ with $q < p$ not reducible to a redex of $\mathcal{B}$, and $t|_p$ transfinitely reducible to a term in $\mathcal{U}$. Hence, by definition of $\omega_{\mathcal{U}}$, we have $p \in \mathcal{P}os(\omega_{\mathcal{U}}(t))$ and $\omega_{\mathcal{U}}(t)|_p = \bot$, which implies $root(\mathrm{BLT}_{\mathcal{U}}(s)|_p) = \bot$. In case $f \neq \bot$, there exists by Lemma 3.10(2) and compression a finite partial term t such that $t|_q$ with $q \leq p$ not reducible to a redex of $\mathcal{B}$. Hence, $root(\omega_{\mathcal{U}}(t)) = f$, which implies $root(\mathrm{BLT}_{\mathcal{U}}(s)|_p) = f$.

Hence, $root(\mathrm{BLT}_{\mathcal{U}}(s)|_p) = f$ iff $root(\mathrm{BLT}^{\infty}_{\mathcal{U}}(s)|_p) = f$, as required. □

4.2 From Direct Approximants to Infinitary Rewriting

Although a Böhm-like tree defined by a direct approximant function exists for every Böhm-like tree defined by a set of meaningless terms, the reverse does not hold. To see this, recall congruence holds for every Böhm-like tree defined by a set of meaningless terms (see Corollary 3.12). Congruence does not necessarily hold for Böhm-like trees defined by direct approximant functions. Consider e.g. the fully-extended, orthogonal CRS consisting of the following two rewrite rules:

$$\begin{aligned} \mathtt{IsEmpty}(\mathtt{nil}) &\to \mathtt{true} \\ \mathtt{IsEmpty}(x : xs) &\to \mathtt{false} \end{aligned}$$

Moreover, consider the following rules, forming a confluent and terminating CRS:

$$\begin{aligned} \mathtt{IsEmpty}(xs) &\to \bot \\ \mathtt{nil} &\to \bot \end{aligned}$$

The map ω assigning to each term its normal form with respect to the last two rules defines a direct approximant function for the CRS consisting of the first two rules. However, the Böhm-like tree defined by ω is not congruent:

$$\mathrm{BLT}(\bot) = \bot = \mathrm{BLT}(\mathtt{nil})\,,$$

but placed in the context $\mathtt{IsEmpty}(\Box)$:

$$\mathrm{BLT}(\mathtt{IsEmpty}(\bot)) = \mathtt{IsEmpty}(\bot) \neq \mathtt{true} = \mathrm{BLT}(\mathtt{IsEmpty}(\mathtt{nil}))\,.$$

Hence, a class of Böhm-like trees exists that can be defined by means of direct approximant functions, but not by means of a set of meaningless terms.

In the remainder we consider two Böhm-like trees defined by direct approximant functions which we have sets of meaningless terms that do define the same Böhm-like trees: the Berarducci-like trees and the Huet-Lévy trees.

Berarducci-Like Trees. Define $\omega_{\mathrm{BeL}}(s)$ as the largest term t with respect to $\preccurlyeq$ such that $t \preccurlyeq s[\bot]_p$ iff the subterm at position p in s reduces to a redex. Given a fully-extended, orthogonal CRS, it is easily shown that ω_{BeL} defines a direct approximant function; the one associated with Berarducci-like trees.

We show for every fully-extended, orthogonal CRS $\mathcal{C} = (\Sigma, R)$ that its Berarducci-like tree and the Böhm-like tree defined by the set of root-active terms (see Sect. 3.5) coincide for every finite partial term.

Denote the set of terms each of which has a $\bot$-instance that is a root-active term by $\mathcal{U}_{\mathrm{BeL}}$. Moreover, denote by $\mathrm{BLT}^{\infty}_{\mathrm{BeL}}$ the Böhm-like tree defined by $\mathcal{U}_{\mathrm{BeL}}$ and denote by $\mathrm{BLT}_{\mathrm{BeL}}$ the Berarducci-like tree. We show that $\mathrm{BLT}^{\infty}_{\mathrm{BeL}}$ and $\mathrm{BLT}_{\mathrm{BeL}}$ are identical as maps on the finite partial terms. We start with a lemma:

Lemma 4.7. *Let $\mathcal{U}$ be defined as:*

$$\mathcal{U} = \{s \textit{ is a partial term} \mid s \textit{ either root-active or } s \twoheadrightarrow \bot\}\,.$$

It holds that $\mathcal{U} = \mathcal{U}_{\mathrm{BeL}}$.

Proof. We show $\mathcal{U}_{\mathrm{BeL}} \subseteq \mathcal{U}$ and $\mathcal{U} \subseteq \mathcal{U}_{\mathrm{BeL}}$. Thus, suppose $s \in \mathcal{U}_{\mathrm{BeL}}$. By definition of $\mathcal{U}_{\mathrm{BeL}}$, there exists for s a $\bot$-instance t that is root-active. The subterms replaced by $\bot$ either contribute or do not contribute to t being root-active. In case the subterms contribute, we have by orthogonality that $s \twoheadrightarrow \bot$. In case they do not contribute, we have by orthogonality that s is root-active. Hence, $s \in \mathcal{U}$.

That $\mathcal{U} \subseteq \mathcal{U}_{\mathrm{BeL}}$ follows by orthogonality when we replace each $\bot$ in every term of $\mathcal{U}$ by a closed root-active term. In case no root-active term exists, $\mathcal{U} = \{\bot\}$ and we are done immediately. □

We can now prove:

Theorem 4.8. *If s is a finite partial term, then* $\mathrm{BLT}^{\infty}_{\mathrm{BeL}}(s) = \mathrm{BLT}_{\mathrm{BeL}}(s)$.

Proof. Let $\omega_{\mathcal{U}_{\mathrm{BeL}}}$ be defined according to Definition 4.4, with $\mathcal{U}_{\mathrm{BeL}}$ assuming the rôle of $\mathcal{U}$. By Lemma 4.7, compression, the observation that $\mathcal{U}_{\mathrm{BeL}}$ is closed under transfinite expansion, and Definition 4.4, we have that $\omega_{\mathcal{U}_{\mathrm{BeL}}}$ replaces by $\bot$ precisely every maximal subterm that reduces to a redex — note that $s \twoheadrightarrow \bot$ either has a redex at the root or $s = \bot$. Hence, $\omega_{\mathcal{U}_{\mathrm{BeL}}} = \omega_{\mathrm{BeL}}$ and, by Theorem 4.6, we have for each finite partial term s that $\mathrm{BLT}^{\infty}_{\mathrm{BeL}}(s) = \mathrm{BLT}_{\mathrm{BeL}}(s)$. □

Huet-Lévy Trees. By Proposition 3.21, the Huet-Lévy CRS of a fully-extended, orthogonal CRS $\mathcal{C}$ defines a direct approximant function and, hence, a Böhm-like tree, the Huet-Lévy tree.

Denote by $\mathrm{BLT}^{\infty}_{\mathrm{HL}}$ the Böhm-like tree defined by $\mathcal{U}_{\mathrm{HL}}$ and by $\mathrm{BLT}_{\mathrm{HL}}$ the Huet-Lévy tree. We show that $\mathrm{BLT}^{\infty}_{\mathrm{HL}}$ and $\mathrm{BLT}_{\mathrm{HL}}$ are identical as maps on the finite partial terms.

Theorem 4.9. *If s is a finite partial term, then* $\mathrm{BLT}^{\infty}_{\mathrm{HL}}(s) = \mathrm{BLT}_{\mathrm{HL}}(s)$.

Proof. Suppose s is a finite partial term and let $\omega_{\mathcal{U}_{\mathrm{HL}}}$ be defined according to Definition 4.4, with $\mathcal{U}_{\mathrm{HL}}$ assuming the rôle of $\mathcal{U}$. By definition of $\omega_{\mathcal{U}_{\mathrm{HL}}}$, the subterms of s that either transfinitely reduce to a redex or term in $\mathcal{U}_{\mathrm{HL}}$ are replaced by $\bot$. Hence, by definition of $\mathcal{U}_{\mathrm{HL}}$, all replaced subterms of s have $\bot$ as their Huet-Lévy direct approximant.

If $\omega_{\mathcal{U}_{\mathrm{HL}}}(s)$ does not replace a certain subterm by $\bot$, then the subterm does not reduce to a redex. Moreover, by definition of $\mathcal{U}_{\mathrm{HL}}$ the subterm reduces in a finite number of steps to a term with a Huet-Lévy direct approximant unequal to $\bot$. Hence, by orthogonality there exists a term t and a reduction $s \to^* t$ such that $\omega_{\mathcal{U}_{\mathrm{HL}}}(s) \preccurlyeq \omega_{\mathrm{HL}}(t)$.

By the facts from the first paragraph and by orthogonality of the Huet-Lévy CRS, we also have $\omega_{\mathrm{HL}}(s) \preccurlyeq \omega_{\mathcal{U}_{\mathrm{HL}}}(s)$. Hence, $\mathrm{BLT}_{\mathcal{U}_{\mathrm{HL}}}(s) = \mathrm{BLT}_{\mathrm{HL}}(s)$ and by Theorem 4.6 we obtain $\mathrm{BLT}^{\infty}_{\mathrm{HL}}(s) = \mathrm{BLT}_{\mathrm{HL}}(s)$. □

5 Conclusion

Somewhat remarkably, there is a difference between the infinitary rewriting approach to Böhm-like trees and the direct approximant approach: Each Böhm-like tree defined by infinitary rewriting coincides with a Böhm-like tree defined by a direct approximant function but the reverse is not the case. The difference seems to be due to the infinitary rewriting approach yielding congruent Böhm-like trees (see Corollary 3.12).

To enable our comparison, we extended to iCRSs the infinitary rewriting approach to Böhm-like trees. Contrary to most of the previous theory developed for iCRSs, no serious complications arise due to iCRSs being higher-order. However, as noted by Van Oostrom (private communication), a number of reasonable Böhm-like trees cannot be defined due to the overlap axiom (see Remark 3.13).

At least two questions remain: First, can either the infinitary rewriting approach be extended or the direct approximant approach be restricted as to obtain coincidence between the two approaches? Second, can the overlap axiom be replaced by some new axiom as to allow certain forms of overlap?

Acknowledgments. The author wishes to thank Jan-Willem Klop, Vincent van Oostrom, Yoshihito Toyama, and Jaco van de Pol for their support.

References

1. Ketema, J.: Böhm-Like Trees for Rewriting. PhD thesis, Vrije Universiteit, Amsterdam (2001)
2. Lévy, J.J.: Réductions correctes et optimales dans le lambda-calcul. PhD thesis, Université de Paris VII (1978)
3. Barendregt, H.P.: The Lambda Calculus: Its Syntax and Semantics, revised edn. Elsevier Science, Amsterdam (1985)
4. Lévy, J.J.: An algebraic interpretation of the $\lambda\beta$K-calculus and the labelled λ-calculus. In: Böhm, C. (ed.) Lambda-Calculus and Computer Science Theory. LNCS, vol. 37, pp. 147–165. Springer, Heidelberg (1975)
5. Longo, G.: Set-theoretical models of λ-calculus: Theories, expansions, isomorphisms. Annals of Pure and Applied Logic 24, 153–188 (1983)
6. Berarducci, A.: Infinite λ-calculus and non-sensible models. In: Logic and Algebra. Lect. Notes Pure Appl. Math., vol. 180, pp. 339–378 (1996)
7. Kennaway, R., van Oostrom, V., de Vries, F.J.: Meaningless terms in rewriting. JFLP 1 (1999)
8. Kennaway, R., del Vries, F.-J.: Infinitary rewriting. In: [18], ch. 12
9. Blom, S.C.C.: Term Graph Rewriting: syntax and semantics. PhD thesis, Vrije Universiteit, Amsterdam (2001)
10. Ketema, J.: Böhm-like trees for term rewriting systems. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 233–248. Springer, Heidelberg (2004)
11. Ketema, J., Simonsen, J.G.: Infinitary combinatory reduction systems. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 438–452. Springer, Heidelberg (2005)
12. Ketema, J., Simonsen, J.G.: On confluence of infinitary combinatory reduction systems. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 199–214. Springer, Heidelberg (2005)
13. Ketema, J., Simonsen, J.G.: Infinitary combinatory reduction systems. Technical Report D-558, Department of Computer Science, University of Copenhagen (2006)
14. Klop, J.W., van Oostrom, V., van Raamsdonk, F.: Combinatory reduction systems: introduction and survey. TCS 121(1-2), 279–308 (1993)
15. Ketema, J.: On normalisation of infinitary combinatory reduction systems. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 172–186. Springer, Heidelberg (2008)
16. Kennaway, R., et al.: Infinitary rewriting: From syntax to semantics. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity. LNCS, vol. 3838, pp. 148–172. Springer, Heidelberg (2005)
17. Kennaway, J.R., et al.: Infinitary lambda calculus. TCS 175(1), 93–125 (1997)
18. Terese (ed.): Term Rewriting Systems. Cambridge University Press, Cambridge (2003)

The Derivational Complexity Induced by the Dependency Pair Method*

Georg Moser and Andreas Schnabl

Institute of Computer Science, University of Innsbruck, Austria
{georg.moser,andreas.schnabl}@uibk.ac.at

Abstract. We study the derivational complexity induced by the (basic) dependency pair method. Suppose the derivational complexity induced by a termination method is closed under elementary functions. We show that the derivational complexity induced by the dependency pair method based on this termination technique is the same as for the direct technique. Therefore, the derivational complexity induced by the dependency pair method based on lexicographic path orders or multiset path orders is multiple recursive or primitive recursive, respectively. Moreover for the dependency pair method based on Knuth-Bendix orders, we obtain that the derivational complexity function is majorised by the Ackermann function. These characterisations are essentially optimal.

1 Introduction

In order to assess the complexity of a terminating term rewrite system (TRS for short) it is natural to look at the maximal length of derivation sequences, as suggested by Hofbauer and Lautemann in [1]. More precisely, the *derivational complexity function* with respect to a terminating TRS $\mathcal{R}$ relates the length of the longest derivation sequence to the size of the initial term. For direct termination methods a considerable number of results establish essentially optimal upper bounds on the growth rate of the derivational complexity function. See e.g. [2,3] for recent results in this direction. However, for transformation techniques like semantic labelling [4] or the dependency pair method [5] the situation changes drastically. Apart from the trivial case of labelling with finite models, only partial results are known. With respect to semantic labelling, [6] establishes bounds on the derivation length of TRS, when natural numbers are used as labels and termination is shown via the Knuth-Bendix order (KBO). And recently in [7,8] the derivation length induced by the basic dependency pair method is investigated. Still in both cases only restricted variants of semantic labelling or the dependency pair method could be analysed, compare [6,7,8].

In this paper we investigate the derivational complexity induced by the basic dependency pair method based on reasonably strong base orders. Suppose the class of derivational complexity functions induced by a direct termination method is closed under elementary functions. Then we show that the derivational

* This research is partly supported by FWF (Austrian Science Fund) project P20133.

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 255–269, 2009.
© Springer-Verlag Berlin Heidelberg 2009

complexity induced by the dependency pair method based on this termination technique is the same as for the direct technique. More precisely we show that the derivational complexity of a TRS whose termination is established via the dependency pair method combined with some base order is triple exponential in the derivational complexity induced by the base order directly. Moreover, we present an example which shows that at least two of the three exponentials in our upper bound can actually be reached.

It should be emphasised that the notion of dependency pair method studied here amounts to the original technique as introduced by Arts and Giesl [5] (see also [9]). Consider the following TRS $\mathcal{R}_1$ taken from [10]:

$$\begin{array}{lll} 1\colon & (x \times y) \times z \to x \times (y \times z) & 3\colon\ (x + y) \times z \to (x \times z) + (y \times z) \\ 2\colon & z \times (x + \mathsf{f}(y)) \to \mathsf{g}(z, y) \times (x + \mathsf{a}) & \end{array}$$

Due to rule 2, termination of $\mathcal{R}_1$ cannot be concluded by the lexicographic path order (LPO), cf. [10]. On the other hand, termination follows easily by the dependency pair method based on LPO, if we use argument filtering.

The gist of our result is that for this standard application of the dependency pair method the derivational complexity induced by LPO directly (which is multiple recursive, cf. [11]) bounds the derivation lengths admitted by the investigated TRS $\mathcal{R}_1$. From this we can conclude that the derivational complexity function of $\mathcal{R}_1$ is multiple recursive. Analogous results hold if we employ the multiset path order (MPO) or KBO as base order. Moreover the thus obtained upper bounds are still tight, which essentially follows from the tight characterisation of the derivational complexity by the indicated base orders [11,12,13].

Note the challenges of such an investigation: In order to estimate the derivation length of $\mathcal{R}_1$ we only consider the derivation length induced by the base order. This implies that we use an upper bound on the maximal number of dependency pair steps to bound the length of derivations. It remains open to what extent such a result holds in general, i.e., beyond the basic dependency pair method. The challenge of such an endeavour is most prominent if we allow an iterative use of the dependency pair transformation as for example in the recursive SCC algorithm (see [9]) or the dependency pair framework (see [14]). It is well-known that two iterations of the recursive SCC algorithm based on the subterm criterion (see [15]) suffice to show termination of (the standard formulation of) the Ackermann function. See also [16] for a like minded example. Clearly in this context a triple exponential function is by far not sufficient to bound the difference between the derivational complexity of the TRS and the derivational complexity induced by the base method directly.

The rest of this paper is organised as follows. In Section 2 we present basic notions and starting points of the paper. Section 3 introduces suitable notions to trace an *implicit dependency pair derivation* in a given derivation over a TRS. Our main result is proved in Section 4, while Section 5 presents the above mentioned example on the lower bound. Finally, we conclude in Section 6.

2 Dependency Pairs

We assume familiarity with the basics of term rewriting, see [17,18]. Below we recall the bare essentials of the basic dependency pair method as put forward in [5], but at least nodding acquaintance with [5] or [9] will prove helpful.

Let $\mathcal{V}$ denote a countably infinite set of variables and $\mathcal{F}$ a signature. The set of terms over $\mathcal{F}$ and $\mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F},\mathcal{V})$. The (proper) subterm relation is denoted as $\trianglelefteq$ ($\triangleleft$). The *root symbol* (denoted as $\mathsf{rt}(t)$) of a term t is either t itself, if $t \in \mathcal{V}$, or the symbol f, if $t = f(t_1,\ldots,t_n)$. The set of *positions* $\mathcal{P}\mathsf{os}(t)$ of a term t is defined as usual. We write $p \leqslant q$ ($p < q$) to denote that p is a (proper) prefix of q, and $p \parallel q$ if neither $p \leqslant q$ nor $q \leqslant p$. The subterm of t at position p is denoted as $t|_p$. We write $\mathcal{P}\mathsf{os}_{\mathcal{F}}(t)$ ($\mathcal{P}\mathsf{os}_{\mathcal{V}}(t)$) for the set of positions p such that $\mathcal{F}$ ($\mathcal{V}$) contains $\mathsf{rt}(t_p)$. The *size* $|t|$ and the *depth* $\mathsf{dp}(t)$ of a term t are defined as usual (e.g., $|\mathsf{f}(\mathsf{a},x)| = 3$ and $\mathsf{dp}(\mathsf{f}(\mathsf{a},x)) = 1$). To simplify the exposition, we often confuse terms and their tree representations. I.e., we call a maximal set of positions B in a term t such that for no $q,q' \in B$, we have $q \parallel q'$, a *branch* of t.

Let $\mathcal{R}$ be a finite TRS over $\mathcal{F}$. We write $\to_{\mathcal{R}}$ (or simply $\to$) for the induced rewrite relation. If we wish to indicate the redex position p and the applied rewrite rule $l \to r$ in a reduction from s to t, we write $s \to_{p,l\to r} t$. The set of defined function symbols is denoted as $\mathcal{D}$, while the constructor symbols are collected in $\mathcal{C}$. The n-fold composition of $\to$ is denoted as $\to^n$ and the *derivation length* of a term s with respect to a finitely branching, well-founded binary relation $\to$ on terms is defined as $\mathsf{dl}(s,\to) := \max\{n \mid \exists t\ s \to^n t\}$. The *derivational complexity function* of $\mathcal{R}$ is defined as: $\mathsf{dc}_{\mathcal{R}}(n) = \max\{\mathsf{dl}(t,\to_{\mathcal{R}}) \mid |t| \leqslant n\}$.

In analogy to $\mathsf{dc}_{\mathcal{R}}$ we define functions tracing the depth or size. The *potential depth* of a term s with respect to $\to$ is defined as follows: $\mathsf{pdp}(s,\to) := \max\{\mathsf{dp}(t) \mid s \to^* t\}$ and the induced *depth growth function* (with respect to $\mathcal{R}$) is defined as $\mathsf{dpg}_{\mathcal{R}}(n) := \max\{\mathsf{pdp}(t,\to_{\mathcal{R}}) \mid |t| \leqslant n\}$. The *potential size* $\mathsf{psz}(s,\to)$ of a term s and the *size growth function* $\mathsf{szg}_{\mathcal{R}}(n)$ are defined similarly.

We recall the central notions of the dependency pair method, see [5,9]. Let t be a term. We set $t^\sharp := t$ if $t \in \mathcal{V}$, and $t^\sharp := f^\sharp(t_1,\ldots,t_n)$ if $t = f(t_1,\ldots,t_n)$. Here $f^\sharp$ is a new n-ary function symbol called *dependency pair symbol*. For a signature $\mathcal{F}$, we define $\mathcal{F}^\sharp = \mathcal{F} \cup \{f^\sharp \mid f \in \mathcal{F}\}$. The set $\mathsf{DP}(\mathcal{R})$ of *dependency pairs* of a TRS $\mathcal{R}$ is defined as $\{l^\sharp \to u^\sharp \mid l \to r \in \mathcal{R}, u \trianglelefteq r, \mathsf{rt}(u) \in \mathcal{D}, l \not\trianglerighteq u\}$.

Proposition 1 ([5,9]). *A TRS $\mathcal{R}$ is terminating if and only if there exists no infinite derivation of the form $t_1^\sharp \to_{\mathcal{R}}^* t_2^\sharp \to_{\mathsf{DP}(\mathcal{R})} t_3^\sharp \to_{\mathcal{R}}^* \ldots$ such that for all $i > 0$, $t_i^\sharp$ is terminating with respect to $\mathcal{R}$.*

Proposition 1 gives rise to the *dependency pair complexity function*:

$$\mathsf{DPc}_{\mathcal{R}}(n) := \max\{\mathsf{dl}(t^\sharp,\to_{\mathsf{DP}(\mathcal{R})/\mathcal{R}}) \mid |t| \leqslant n\}\ ,$$

where we write $\to_{\mathsf{DP}(\mathcal{R})/\mathcal{R}}$ for $\to_{\mathcal{R}}^* \cdot \to_{\mathsf{DP}(\mathcal{R})} \cdot \to_{\mathcal{R}}^*$, cf. [19]. Now, we fix the notion of *basic* dependency pair method. An *argument filtering* (for a signature $\mathcal{F}$) is a mapping π that assigns to every n-ary function symbol $f \in \mathcal{F}$ an argument position $i \in \{1,\ldots,n\}$ or a (possibly empty) list $[i_1,\ldots,i_m]$ of argument

positions with $1 \leqslant i_1 < \cdots < i_m \leqslant n$. The signature $\mathcal{F}_\pi$ consists of all function symbols f such that $\pi(f)$ is some list $[i_1, \ldots, i_m]$, where in $\mathcal{F}_\pi$ the arity of f is m. Every argument filtering π induces a mapping from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}_\pi, \mathcal{V})$, also denoted by π:

$$\pi(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ \pi(t_i) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } \pi(f) = i \\ f(\pi(t_{i_1}), \ldots, \pi(t_{i_m})) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } \pi(f) = [i_1, \ldots, i_m] \end{cases}$$

A *reduction pair* $(\gtrsim, \succ)$ consists of a rewrite preorder $\gtrsim$ and a compatible well-founded order $\succ$ which is closed under substitutions. Here compatibility means the inclusion $\gtrsim \cdot \succ \cdot \gtrsim \subseteq \succ$.

Proposition 2 ([5,9]). *A TRS $\mathcal{R}$ is terminating if and only if there exist an argument filtering π and a reduction pair $(\gtrsim, \succ)$ such that $\pi(\mathsf{DP}(\mathcal{R})) \subseteq \succ$ and $\pi(\mathcal{R}) \subseteq \gtrsim$.*

Let $\mathcal{R}$ be a terminating TRS. In the sequel we show that the derivational complexity function $\mathsf{dc}_\mathcal{R}$ is bounded triple exponentially in the dependency pair complexity function $\mathsf{DPc}_\mathcal{R}$. For that we mainly bound the depth growth function of $\mathcal{R}$ exponentially in $\mathsf{DPc}_\mathcal{R}$. As the maximal length of a nonlooping derivation is exponentially bounded in the size of the occurring terms and the latter is exponentially bounded in their depth, our result then follows. In order to prove the main step we analyse the shape of a potential derivation over $\mathcal{R} \cup \mathsf{DP}(\mathcal{R})$ in the light of a given $\mathcal{R}$-derivation. This is the purpose of the next section.

3 Progenitor and Progeny

We introduce a specific generalisation of the notion of *descendant* of a position p which we call *progeny*. Recall the definition of descendants (see [18, Chapter 4]). Let $A : s \to_{p',l\to r} t$ be a rewriting step, and let $p \in \mathcal{P}\mathsf{os}(s)$. Then the *descendants of p in t* (denoted by $p \setminus A$) are defined as follows:

$$p \setminus A = \begin{cases} \{p\} & \text{if } p < p' \text{ or } p \parallel p', \\ \{p'q_3q_2 \mid r|_{q_3} = l|_{q_1}\} & \text{if } p = p'q_1q_2 \text{ with } q_1 \in \mathcal{P}\mathsf{os}_\mathcal{V}(l), \\ \varnothing & \text{otherwise} \end{cases}$$

In our situation, we also want to keep track of redex positions, not just of positions in the context or the substitution of the rewrite rule. This intuition is cast into the following definition.

Definition 3. *Let $A : s \to_{p',l\to r} t$ be a rewriting step, and let $p \in \mathcal{P}\mathsf{os}(s)$. Then the* progenies of p in t *(denoted by $p \setminus\!\setminus A$) are:*

$$p \setminus\!\setminus A = \begin{cases} \{p\} & \text{if } p < p' \text{ or } p \parallel p', \\ \{p'q_3q_2 \mid r|_{q_3} = l|_{q_1}\} & \text{if } p = p'q_1q_2 \text{ with } q_1 \in \mathcal{P}\mathsf{os}_\mathcal{V}(l), \\ \{p'q_2 \mid r|_{q_2} = l|_{q_1}\} & \text{if } p = p'q_1 \text{ with } q_1 \in \mathcal{P}\mathsf{os}_\mathcal{F}(l) \setminus \{\epsilon\}, \\ \{pq_1 \mid r|_{q_1} \not\trianglelefteq l \wedge q_1 \in \mathcal{P}\mathsf{os}_\mathcal{F}(r)\} & \text{if } p = p' \end{cases}$$

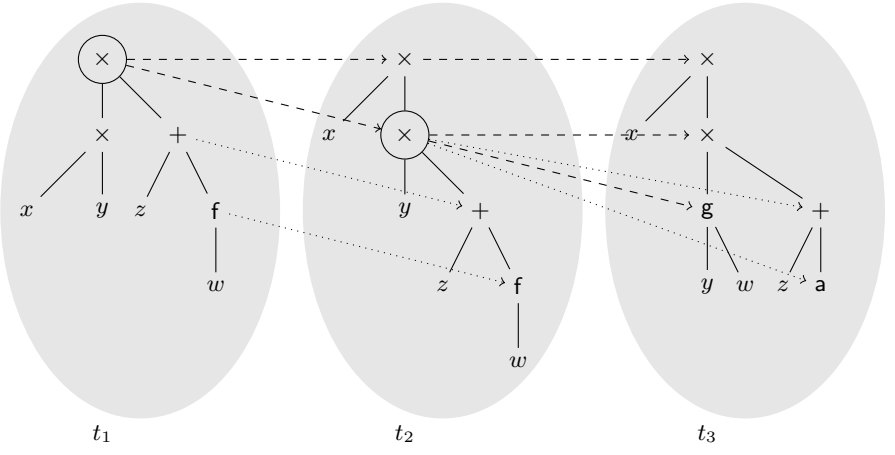


Fig. 1. A derivation, its progeny relation and redex positions

If $q \in p \setminus\!\setminus A$, then we also say that p is a progenitor of q in s. *We denote the set of progenitors of q in s by $A \setminus\!\setminus q$. For a set $P \subseteq \mathcal{P}\mathsf{os}(s)$, we define $P \setminus\!\setminus A = \bigcup_{p \in P} p \setminus\!\setminus A$.*

Note that the distinction between the last two cases corresponds to the exclusion of rules $l^\sharp \to u^\sharp$ from $\mathsf{DP}(\mathcal{R})$ where $u \triangleleft l$, see Section 2.

Example 4. Consider the TRS $\mathcal{R}_1$ from Section 1 and let $t_1 = (x \times y) \times (z + \mathsf{f}(w))$, $t_2 = x \times (y \times (z + \mathsf{f}(w)))$, and $t_3 = x \times (\mathsf{g}(y, w) \times (z + \mathsf{a}))$. We have the derivation $A : t_1 \to t_2 \to t_3$, cf. Figure 1. Redex positions are marked by circles, the progeny relation is marked by dotted and dashed lines (the two kinds of lines will be distinguished in Example 15 below). For clarity progeny relations between variables have been omitted.

Lemma 5. *Let $\mathcal{R}$ be a TRS, let $A : s \to_{\mathcal{R}} t$, let $p \in \mathcal{P}\mathsf{os}(s)$, and let $q \in \mathcal{P}\mathsf{os}(t)$. If $q \in p \setminus\!\setminus A$ and $\mathsf{rt}(t|_q) \in \mathcal{D}$, then $\mathsf{rt}(s|_p) \in \mathcal{D}$ and $(s|_p)^\sharp \to^{=}_{\mathcal{R} \cup \mathsf{DP}(\mathcal{R})} (t|_q)^\sharp$.*

Proof. Suppose that A is $s \to_{p', l \to r} t$. If $p < p'$ or $p \parallel p'$, then by definition, we have $p = q$ and thus $(s|_p)^\sharp \to^{=}_{\mathcal{R}} (t|_q)^\sharp$. On the other hand, if $p = p'$, then there exists $q_1 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$ such that $q = p'q_1$. Moreover, $t|_q \not\triangleleft s|_p$. By assumption $\mathsf{rt}(t|_q) \in \mathcal{D}$ and thus we obtain $(s|_p)^\sharp \to_{\mathsf{DP}(\mathcal{R})} (t|_q)^\sharp$. Finally, if $p > p'$, then by definition, we have $s|_p = t|_q$. Then (trivially) $(s|_p)^\sharp \to^{=}_{\mathcal{R}} (t|_q)^\sharp$. □

Lemma 6. *Let $A : s \to_{p', l \to r} t$ be a rewriting step. Then for every $q \in \mathcal{P}\mathsf{os}(t)$, we have $A \setminus\!\setminus q \neq \varnothing$.*

Proof. If $q < p'$ or $q \parallel p'$, then $A \setminus\!\setminus q = \{q\}$. If $q = p'q_1$, $q_1 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$, and $t|_q \not\triangleleft s|_{p'}$, then $A \setminus\!\setminus q = \{p'\}$. If $q = p'q_1$, $q_1 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$, and $t|_q \triangleleft s|_{p'}$, then there is some p_1 such that $s|_{p'p_1} = t|_q$, so $p'p_1 \in A \setminus\!\setminus q$. Last, if $q = p'q_1q_2$ and $q_1 \in \mathcal{P}\mathsf{os}_{\mathcal{V}}(r)$, then there is some p_1 such that $s|_{p'p_1} = t|_{p'q_1}$. Therefore, $p'p_1q_2 \in A \setminus\!\setminus q$. □

Definition 3 and Lemmata 5 and 6 extend to derivations in the natural way:

Definition 7. *Let $A : s \to^* t$ be a derivation, and let $p \in \mathcal{P}\mathsf{os}(s)$. Then the* progenies of p in t *(also denoted by $p \setminus\!\setminus A$) are defined as follows:*

- *If A is the empty derivation, then $p \setminus\!\setminus A = \{p\}$.*
- *Otherwise, we can split A into $A_1 : s \to s'$ and $A_2 : s' \to^* t$. Then $p \setminus\!\setminus A = (p \setminus\!\setminus A_1) \setminus\!\setminus A_2$.*

We say p is a progenitor *of q if $p \in A \setminus\!\setminus q$, which holds if $q \in p \setminus\!\setminus A$. Moreover, we have $q \in P \setminus\!\setminus A$ if and only if $q \in p \setminus\!\setminus A$ for some $p \in P$.*

The next lemma follows by straightforward induction using Lemmata 6 and 5.

Lemma 8. *Let $A : s \to^* t$ be a derivation, and let $p \in \mathcal{P}\mathsf{os}(s)$, $q \in \mathcal{P}\mathsf{os}(t)$. Then the set $A \setminus\!\setminus q$ of progenitors of q is not empty. Moreover if $q \in p \setminus\!\setminus A$ with $\mathsf{rt}(t|_q) \in \mathcal{D}$, then $\mathsf{rt}(s|_p) \in \mathcal{D}$ and $(s|_p)^\sharp \to^*_{\mathcal{R} \cup \mathsf{DP}(\mathcal{R})} (t|_q)^\sharp$.*

Using Lemma 8, we can extract derivations over $\mathcal{R} \cup \mathsf{DP}(\mathcal{R})$ from a given derivation in a TRS $\mathcal{R}$ using positions connected by the progeny relation.

Definition 9. *Let $\mathcal{R}$ be a TRS, let $t_1, \ldots, t_n$ be terms, and let $p_1, \ldots, p_n$ be positions in $t_1, \ldots, t_n$, respectively, such that $\mathsf{rt}(t_n|_{p_n}) \in \mathcal{D}$, and for all $1 \leqslant i \leqslant n-1$, we have $A_i : t_i \to_{\mathcal{R}} t_{i+1}$ and $p_{i+1} \in p_i \setminus\!\setminus A_i$. Then we call $A : (t_1|_{p_1})^\sharp \to^*_{\mathcal{R} \cup \mathsf{DP}(\mathcal{R})} (t_n|_{p_n})^\sharp$ the* implicit dependency pair derivation *with respect to $t_1, \ldots, t_n$ and $p_1, \ldots, p_n$. We denote the number of $\mathsf{DP}(\mathcal{R})$-steps in A as $\mathsf{DPI}(A)$.*

Note that Definition 9 is well-defined, due to Lemma 8.

Example 10 (continued from Example 4). The implicit dependency pair derivation with respect to t_1, t_2, t_3 and $\epsilon, 2, 2$ is given as follows:

$$t_1^\sharp \;\to_{\mathsf{DP}(\mathcal{R}_1)}\; y \times^\sharp (z + \mathsf{f}(w)) \;\to_{\mathsf{DP}(\mathcal{R}_1)}\; \mathsf{g}(y, w) \times^\sharp (z + \mathsf{a})\,.$$

Lemma 11. *Let $A : s \to_{p', l \to r} t$ be a rewriting step. Let $q, q' \in \mathcal{P}\mathsf{os}(t)$. If $q \leqslant q'$, then for any $p_0 \in A \setminus\!\setminus q$, there exists $p_0' \in A \setminus\!\setminus q'$ such that $p_0 \leqslant p_0'$.*

Proof. According to Definition 3, there are four cases for q'.

- If $q' < p'$ or $q' \parallel p'$, then also $q < p'$ or $q \parallel p'$. Therefore, $A \setminus\!\setminus q = \{q\}$ and $A \setminus\!\setminus q' = \{q'\}$.
- If $q' = p'q_1'$, $q_1' \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$, and $r|_{q_1'} \not\trianglelefteq l$, then either $q < p'$, or $q = p'q_1$, $q_1 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$, and $r|_{q_1} \not\trianglelefteq l$. We have $A \setminus\!\setminus q = \{p_0\}$ and $A \setminus\!\setminus q' = \{p'\}$ with $p_0 = q$ or $p_0 = p'$.
- If $q' = p'q_1'$, $q_1' \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$, and $r|_{q_1'} \trianglelefteq l$, then $A \setminus\!\setminus q' = \{p'q_2' \mid q_2' \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(l) \wedge r|_{q_1'} = l|_{q_2'}\}$. Three cases from Definition 3 are applicable for q. Suppose $q < p'$, then $A \setminus\!\setminus q = \{q\}$, and $q < p' \leqslant p'q_2'$ for any $q_2' \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(l)$. If $q = p'q_1$, $q_1 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$, and $r|_{q_1} \not\trianglelefteq l$, then $A \setminus\!\setminus q = \{p'\}$. Last, if $q = p'q_1$, $q_1 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$, and $r|_{q_1} \trianglelefteq l$, then $A \setminus\!\setminus q = \{p'q_2 \mid q_2 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(l) \wedge r|_{q_1} = l|_{q_2}\}$. We have $q' = qq_3'$, so for any $p'q_2 \in A \setminus\!\setminus q$, also $p'q_2q_3' \in A \setminus\!\setminus q'$.

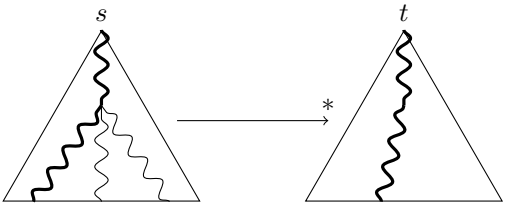
– Otherwise, $q' = p'q'_1q'_2$ with $q'_1 \in \mathcal{P}\mathsf{os}_{\mathcal{V}}(r)$. Then $A \setminus\setminus q' = \{p'q'_3q'_2 \mid r|_{q'_1} = l|_{q'_3}\}$. Except for $q \parallel p'$, all cases in Definition 3 can happen for q. Suppose $q < p'$, then $A \setminus\setminus q = \{q\}$, and $q < p' < p'q'_3q'_2$ for any $q'_3 \in \mathcal{P}\mathsf{os}_{\mathcal{V}}(l)$. If $q = p'q_1$, $q_1 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$, and $r|_{q_1} \not\trianglelefteq l$, then $A \setminus\setminus q = \{p'\}$. For the next case, suppose $q = p'q_1$, $q_1 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$, and $r|_{q_1} \trianglelefteq l$. Then $A \setminus\setminus q = \{p'q_2 \mid q_2 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(l) \wedge r|_{q_1} = l|_{q_2}\}$. We have $q' = qq'_4q'_2$, so for any $p'q_2 \in A \setminus\setminus q$, also $p'q_2q'_4q'_2 \in A \setminus\setminus q'$. Otherwise, $q = p'q'_1q_2$. Then $A \setminus\setminus q = \{p'q_3q_2 \mid r|_{q'_1} = l|_{q_3}\}$. We have $q'_2 = q_2q'_4$, hence for any $p'q_3q_2 \in A \setminus\setminus q$, also $p'q_3q'_2 \in A \setminus\setminus q'$.

□

Note that each position in a term may have several progenitors:

Example 12. Consider the TRS $\mathcal{R}_2$ consisting of the single rule $\mathsf{f}(x,x) \to g(x)$, and the rewrite step $A : \mathsf{f}(\mathsf{0},\mathsf{0}) \to_{\mathcal{R}_2} \mathsf{g}(\mathsf{0})$. Then $A \setminus\setminus 1 = \{1,2\}$.

We restrict the progenies and progenitors to a single branch in each term. The definition rests on the idea that for a derivation $A\colon s \to^* t$ and a main branch B' in t it is possible to find a *main branch* B in s such that each position $q \in B'$ has a (unique) progenitor in B; see the picture below for an illustration:



In the following definition, the restriction to the leftmost of all candidate positions is arbitrary and can be suitably replaced. Note that its second clause is well-defined by Lemmata 8 and 11.

Definition 13. *Let $A\colon t_1 \to^* t_n$ denote a derivation built up from the rewrite steps $A_i\colon t_i \to t_{i+1}$ for $i = 1,\ldots,n-1$. Then the* main branch *of each term in A is inductively defined:*

– *The main branch of t_n is the leftmost branch of maximal length in t_n.*
– *Suppose the main branch of t_{i+1} is denoted as B_{i+1}, $1 \leqslant i \leqslant n-1$. Then consider all branches b in t_i such that for every $q \in B_{i+1}$, the set of progenitors $A_i \setminus\setminus q$ of q has nonempty intersection with b. The leftmost of these branches is the main branch of t_i, denoted as B_i.*

The next definition specialises progenies and progenitors to the main branch.

Definition 14. *Let $A' : s \to t$ be a rewriting step, let $p \in \mathcal{P}\mathsf{os}(s)$, and let B and B' be branches in s and t. Then the set of* main progenies of p in t *(with respect to A') (denoted as $p \supset^B_{B'} A'$) is defined as follows:*

$$p \supset^B_{B'} A' = \begin{cases} \varnothing & \text{if } p \notin B \\ B' \cap p \setminus\setminus A' & \text{if } p \in B \end{cases}$$

We naturally extend this definition to derivations, analogous to Definition 7. If the (main) branches B and B' are clear from context, we write $p \supset A'$ instead of $p \supset^B_{B'} A'$. If $q \in p \supset A'$, then we also say that p is a main progenitor of q in s *(with respect to A'). We denote the set of main progenitors of q in s by $A' \supset q$. For a set $P \subseteq \mathcal{P}\mathsf{os}(s')$, we define $P \supset A' = \bigcup_{p \in P} p \supset A'$.*

Example 15 (continued from Example 4). Consider the derivation A again. The "central" branch of each term in Figure 1 is its main branch, and the dashed lines denote the main progeny relation.

Lemma 16. *Let $A\colon u \to^* s \to^n t \to^* w$ be a derivation, and denote $A'\colon s \to^n t$. Let $B(s)$ ($B(t)$) denote the main branch of s (t) in A. Then for any $q \in B(t)$, the main progenitor of q in the branch $B(s)$ is unique, i.e., $|A' \supset q| = 1$.*

Proof. By Definition 13, q has at least one main progenitor in s. We show that there exists at most one by induction on n. For $n = 0$ the claim is trivial. Hence assume $n > 0$ and let $A'\colon s \to t' \to^{n-1} t$. Let $B(t')$ denote the main branch in t' with respect to A. By induction hypothesis there exists a unique position q_1 in $B(t')$ such that $(t' \to^{n-1} t) \supset q = \{q_1\}$. Let $A''\colon s \to_{p',l \to r} t'$ denote the first rewrite step in A'. Suppose $q_1 < p'$ or $q_1 \parallel p'$. Then by definition $A'' \setminus\!\setminus q_1 = \{q_1\}$. Hence the main progenitor of q in $B(s)$ is unique. On the other hand suppose $q_1 = p'q_2$ with $q_2 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$ such that $r|_{q_2} \not\trianglelefteq l$. Then $A'' \setminus\!\setminus q_1 = \{p'\}$ and $A' \supset q$ is a singleton as it should be. Now suppose $q_1 = p'q_2$ with $q_2 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(r)$ such that $r|_{q_2} \trianglelefteq l$. Then by definition $A'' \setminus\!\setminus q_1 = \{p'p_1 \mid p_1 \in \mathcal{P}\mathsf{os}_{\mathcal{F}}(l) \wedge l|_{p_1} = r|_{q_2}\}$. Note that $A' \supset q = A'' \setminus\!\setminus q_1 \cap B(s)$, which is again a singleton. Finally, if $q_1 = p'q_2q_3$ with $q_2 \in \mathcal{P}\mathsf{os}_{\mathcal{V}}(r)$, then $A'' \setminus\!\setminus q_1 = \{p'p_1q_3 \mid p_1 \in \mathcal{P}\mathsf{os}_{\mathcal{V}}(l) \wedge l|_{p_1} = r|_{q_2}\}$. As before, the intersection of the latter set with $B(s)$ is a singleton. Hence the main progenitor of q in $B(s)$ is unique. This concludes the inductive proof. □

Lemma 17. *We assume the same notation as in Lemma 16. For any $p \in B(s)$ such that $\mathsf{rt}(s|_p) \in \mathcal{C} \cup \mathcal{V}$, we have $|p \supset A'| \leqslant 1$, i.e., the number of main progenies for a position, whose root is non-defined is at most* 1.

Proof. By induction on n. It suffices to consider the case $n > 0$, so $A'\colon s \to t' \to^{n-1} t$. Let $A''\colon s \to_{p',l \to r} t'$ denote the first rewrite step in A'. If $p < p'$ or $p \parallel p'$, then $p \supset A'' = \{p\}$. If $p > p'$, then for any $q_1 \in p \setminus\!\setminus A''$, we have $s|_p = t'|_{q_1}$, so again, $p \setminus\!\setminus A'' \cap B(t')$ is a singleton. In all three cases, the claim follows by induction hypothesis as $\mathsf{rt}(s|_p) = \mathsf{rt}(t|_{q_1})$. This concludes the proof, as the case $p = p'$ is impossible. Otherwise, we derive a contradiction to the assumption that the root of $s|_p$ is not a defined symbol. □

4 Dependency Pairs and Complexity

In this section, we relate the dependency pair complexity and the derivational complexity of a TRS. As mentioned above the main step is to show that the depth growth of a TRS is bounded by a single exponential in its dependency pair complexity. In the sequel, we fix a finite TRS $\mathcal{R}$ and a derivation $A : t_1 \to^* t_n$

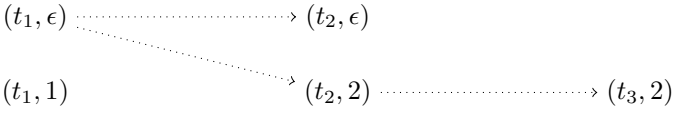
over $\mathcal{R}$ such that $B_1, \ldots, B_n$ denote the main branches with respect to A. We can view the main progeny relation as a graph, called *progenitor graph*. The nodes of a progenitor graph are pairs (t_i, p) representing positions p in terms in A that are directly affected by dependency pair steps. Each edge corresponds to a dependency pair step (and possibly a number of $\mathcal{R}$-steps) in an implicit dependency pair derivation. Each connected component of the progenitor graph is a tree whose height is bounded by the number of dependency pair steps. Using the exponential relationship between the height of this tree and its number of leaves, we bound the depth of the final term in A exponentially in the length of the largest implicit dependency pair derivation, entailing our main result.

Definition 18. *The* progenitor graph *of A is defined as follows.*

- *The nodes are all pairs (t_i, p) such that $p \in B_i$, $\mathsf{rt}(t_i|_p) \in \mathcal{D}$, and either $i = 1$ or the single element of $(t_{i-1} \to t_i) \supset p$ and the redex position in the rewrite step $t_{i-1} \to t_i$ coincide.*
- *There is an edge from (t_i, p) to (t_j, q) whenever $i < j$, $(t_i \to^* t_j) \supset q = \{p\}$, and for all $i \leqslant k < j - 1$, the single element of $(t_k \to^* t_j) \supset q$ and the redex position in the rewrite step $t_k \to t_{k+1}$ do not coincide.*

Note that, due to the definition of the set of nodes in a progenitor graph, the single element of $(t_{j-1} \to t_j) \supset q$ and the redex position in the rewrite step $t_{j-1} \to t_j$ do coincide in the second clause of Definition 18.

Example 19. Consider the derivation A from Example 4 again. Its progenitor graph is shown below:



Lemma 20. *If there is an edge from (t_i, p) to (t_j, q) in G, then there is a derivation $(t_i|_p)^\sharp \to^*_{\mathcal{R}} (t_{j-1}|_{p_1})^\sharp \to_{\mathsf{DP}(\mathcal{R})} (t_j|_q)^\sharp$.*

Proof. By definition, $q \in p \supset (t_i \to^* t_j)$. Therefore, by Lemma 8, we have the implicit dependency pair derivation $A' : (t_i|_p)^\sharp \to^*_{\mathcal{R} \cup \mathsf{DP}(\mathcal{R})} (t_j|_q)^\sharp$. We have $(t_{j-1} \to t_j) \supset q = \{p_1\}$, where by definition p_1 is the redex position of the step $t_{j-1} \to t_j$. Therefore, the last step of A' is a $\mathsf{DP}(\mathcal{R})$-step (see also the last clause of Definition 3). Note that for $i \leqslant k < j - 1$, the single element of $(t_k \to^* t_j) \supset q$ and the redex position in $t_k \to t_{k+1}$ do not coincide. Hence, if there are rewrite steps before the last step, these are $\mathcal{R}$-steps and the lemma follows. □

The next lemma shows, when specialised to the conditions in the first clause of Definition 18, that only nodes which do not contribute to the branching of the progenitor graph, are left out by the definition.

Lemma 21. *Let $p \in B_i$ and $q \in B_j$ such that $i < j$ and $(t_i \to^* t_j) \supset q = \{p\}$. If for all $i \leqslant k \leqslant j - 1$, the single element of $(t_k \to^* t_j) \supset q$ and the redex position in the rewrite step $t_k \to t_{k+1}$ do not coincide, then $p \supset (t_i \to^* t_j) = \{q\}$.*

Proof. We show the lemma by induction on $j-i$. If $i=j$ then the claim trivially holds. Otherwise, the derivation $t_i \to^* t_j$ can be split to $t_i \to t_{i+1} \to^* t_j$. Let p' be the redex position in $t_i \to t_{i+1}$. If $p \parallel p'$, $p < p'$, or $p > p'$, then as in Lemma 17, $|p \supset (t_i \to t_{i+1})| \leqslant 1$, and the lemma follows by induction hypothesis. The last case is again impossible, since by assumption, p and p' do not coincide. □

From now on, let G be the progenitor graph of A. In the next lemmata, we show the properties which allow us to bound $\mathsf{dp}(t_n)$ in the height of G. First, we prove that almost each position in B_n is "covered" by a node in G. Next, we show that each node in G can only cover c positions in B_n, and finally, we show that the branching factor of G is at most c, where $c := \max\{2\} \cup \{\mathsf{dp}(r) \mid l \to r \in \mathcal{R}\}$.

Lemma 22. *For every $q \in B_n$, there either exists $p \in B_1$ such that $\mathsf{rt}(t_1|_p) \in \mathcal{C} \cup \mathcal{V}$ and $A \supset q = \{p\}$, or there exists a node (t_i, p) in G where $q \in p \supset (t_i \to^* t_n)$ and for any successor node (t_j, p_1) of (t_i, p) in G, we have $q \notin p_1 \supset (t_j \to^* t_n)$.*

Proof. By Lemma 16, $A \supset q = \{p\}$ for some $p \in B_1$. If $\mathsf{rt}(t_1|_p) \in \mathcal{C} \cup \mathcal{V}$, the first alternative of the lemma holds. If $\mathsf{rt}(t_1|_p) \in \mathcal{D}$, then $(t_1, p) \in G$. Therefore, there exists a maximal natural number k such that $(t_k, p_2) \in G$ and $q \in p_2 \supset (t_k \to^* t_n)$ for some $p_2 \in B_k$, so the second alternative of the lemma holds for (t_k, p_2). □

Lemma 23. *For every node (t_i, p) in G, there are at most c many positions $q \in B_n$ such that $q \in p \supset (t_i \to^* t_n)$, but for any successor node (t_j, p_1) of (t_i, p), we have $q \notin p_1 \supset (t_j \to^* t_n)$.*

Proof. If there is no $i \leqslant k < n$ such that the redex position of the step $t_k \to t_{k+1}$ and an element of $p \supset (t_i \to^* t_k)$ coincide, then it follows from Lemma 21 that $|p \supset (t_i \to^* t_n)| \leqslant 1$. Otherwise, let k be the smallest number such that $k \geqslant i$ and $p \supset (t_i \to^* t_k) = \{p_2\}$, where p_2 is the redex position of $t_k \to t_{k+1}$. By Definitions 3 and 14, $|p_2 \supset (t_k \to t_{k+1})| \leqslant c$. For each $p_3 \in p_2 \supset (t_k \to t_{k+1})$, if $\mathsf{rt}(t_{k+1}|_{p_3}) \in \mathcal{D}$, then (t_{k+1}, p_3) is a successor node of (t_i, p), and the condition $q \notin p_3 \supset (t_{k+1} \to^* t_n)$ is violated for any main progeny q of p_3. On the other hand, if $\mathsf{rt}(t_{k+1}|_{p_3}) \in \mathcal{C} \cup \mathcal{V}$, then by Lemma 17, $|p_3 \supset (t_{k+1} \to^* t_n)| \leqslant 1$. Thus, in total, there are at most c many elements in $p \supset (t_i \to^* t_n)$ meeting our assumption. □

The following example illustrates the role of Lemma 23.

Example 24. Let $\mathcal{R}_3$ be the TRS consisting of the single rewrite rule

$$\mathsf{d}(\mathsf{S}(x)) \to \mathsf{S}(\mathsf{S}(\mathsf{d}(x))) \ .$$

Let $t_1 = \mathsf{d}(\mathsf{S}(\mathsf{S}(\mathsf{0})))$, $t_2 = \mathsf{S}(\mathsf{S}(\mathsf{d}(\mathsf{S}(\mathsf{0}))))$, and $t_3 = \mathsf{S}(\mathsf{S}(\mathsf{S}(\mathsf{S}(\mathsf{d}(\mathsf{0}))))))$. We have the derivation $A : t_1 \to t_2 \to t_3$ and the following progenitor graph:

Note that G leaves out all function symbols S above the d in each term. However, by Lemma 23, the number of positions in the last term of A which are hidden in this way is only linear in the size of the progenitor graph.

Lemma 25. *Every node in G has at most c many successor nodes.*

Proof. Let (t_i, p) be a node in G. If there is no $i \leqslant j < n$ such that the redex position of the step $t_j \to t_{j+1}$ and an element of $p \supsetneq (t_i \to^* t_j)$ coincide, then (t_i, p) has no successor node, so the claim holds. Otherwise, let j be the smallest number greater than i such that $p \supsetneq (t_i \to^* t_j) = \{q\}$, where q is the redex position of $t_j \to t_{j+1}$. By Definitions 3 and 14, $|q \supsetneq (t_j \to t_{j+1})| \leqslant c$. Hence, (t_i, p) has at most c many successor nodes. □

Now we are ready to prove our main lemma.

Lemma 26. *For every finite and terminating TRS $\mathcal{R}$, there exists a constant C such that for all terms s, we have* $\mathsf{pdp}(s, \to_{\mathcal{R}}) \leqslant |s| \cdot 2^{C \cdot \max\{\mathsf{dl}((s')^\sharp, \to_{\mathsf{DP}(\mathcal{R})/\mathcal{R}}) \mid s' \trianglelefteq s\}}$.

Proof. We show the lemma by proving that for any derivation $A : s \to^*_{\mathcal{R}} t$, there exists a derivation $A' : (s')^\sharp \to^*_{\mathcal{R} \cup \mathsf{DP}(\mathcal{R})} (t')^\sharp$ with $s' \trianglelefteq s$ and $\mathsf{dp}(t) \leqslant |s| \cdot c^{\mathsf{DPI}(A')+2}$ (recall $c = \max\{2\} \cup \{\mathsf{dp}(r) \mid l \to r \in \mathcal{R}\}$). Let k be the number of defined symbols in the main branch of s. The main branch of t consists of $\mathsf{dp}(t) + 1$ many positions, all of which have to fulfil one of the two properties outlined in Lemma 22. By Lemma 17, the first case applies to at most $\mathsf{dp}(s) + 1 - k$ many positions, so for the $\mathsf{dp}(t) - \mathsf{dp}(s) + k$ other positions, the second case applies. By Lemma 23, each node in the progenitor graph G of A can cover at most c many of those positions, so G has to contain at least $\frac{\mathsf{dp}(t) - \mathsf{dp}(s) + k}{c}$ many nodes. There are k many connected components (trees) in G, hence the largest one of them contains at least

$$\frac{\mathsf{dp}(t) - \mathsf{dp}(s) + k}{kc} ,$$

many nodes. Let d be the smallest natural number such that

$$\frac{\mathsf{dp}(t) - \mathsf{dp}(s) + k}{kc} \leqslant c^{d-1} .$$

By Lemma 25, this means that there exists a leaf in the largest tree of G whose distance from the root is at least $d - 2$: recall that any c-ary tree of height $d - 3$ has at most $\frac{c^{d-2}-1}{c-1} \leqslant c^{d-2}$ many nodes (here the height of a tree is the number of edges on the longest path from the root to a leaf). Moreover, by Lemma 20, this path in the graph induces a derivation $A' : (s')^\sharp \to^*_{\mathcal{R} \cup \mathsf{DP}(\mathcal{R})} (t')^\sharp$ with $\mathsf{DPI}(A') \geqslant d - 2$ and $s \trianglerighteq s'$. Reformulating the inequality above yields

$$\mathsf{dp}(t) \leqslant k \cdot c^d + \mathsf{dp}(s) - k \leqslant (\mathsf{dp}(s) + 1) \cdot c^d \leqslant |s| \cdot c^d ,$$

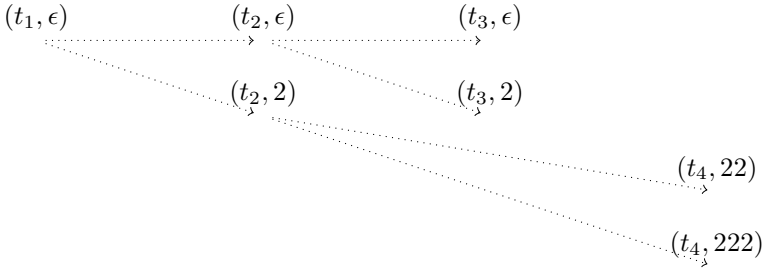
so A' is indeed the derivation we are looking for. □

The main factor of the faster growth of $\mathsf{dp}(t_n)$ compared to the height of G is the difference between the height and the size of G. This becomes apparent in our next example, where G is a full binary tree.

Example 27. Consider the TRS $\mathcal{R}_4$ consisting of the single rewrite rule

$$\mathsf{f}(\mathsf{S}(x), y) \to \mathsf{f}(x, \mathsf{f}(x, y)) \ .$$

Let $t_1 = \mathsf{f}(\mathsf{S}(\mathsf{S}(0)), 0)$, $t_2 = \mathsf{f}(\mathsf{S}(0), \mathsf{f}(\mathsf{S}(0), 0))$, $t_3 = \mathsf{f}(0, \mathsf{f}(0, \mathsf{f}(\mathsf{S}(0), 0)))$, and $t_4 = \mathsf{f}(0, \mathsf{f}(0, \mathsf{f}(0, \mathsf{f}(0, 0))))$. We have the derivation $A : t_1 \to t_2 \to t_3 \to t_4$. The progenitor graph of A is shown below.



Perhaps counter-intuitively in the context of the dependency pair method, the connected component of G with the greatest height need not be the component with the root (t_1, ϵ). All that is left to show is that the derivational complexity of a finite and terminating TRS is bounded double exponentially in its depth growth. This can be achieved by two easy observations.

Lemma 28. *Let $\mathcal{R}$ be a finite and terminating TRS. Then there exists a constant C such that for every term t, we have*

$$\mathsf{dl}(t, \to_{\mathcal{R}}) \leqslant 2^{2^{C \cdot \mathsf{pdp}(t, \to_{\mathcal{R}})}}$$

Proof. We show that there exist constants D and E, such that for all terms t, the inequalities $\mathsf{psz}(t, \to_{\mathcal{R}}) \leqslant 2^{D \cdot \mathsf{pdp}(t, \to_{\mathcal{R}})}$ and $\mathsf{dl}(t, \to_{\mathcal{R}}) \leqslant 2^{E \cdot \mathsf{psz}(t, \to_{\mathcal{R}})}$ hold.

1. For any term t, we have $|t| \leqslant k^{\mathsf{dp}(t)+1}$, where k is the maximum arity of any function symbol in the signature. This proves the first inequality.
2. On the other hand, by assumption the signature $\mathcal{F}$ of $\mathcal{R}$ is finite. Moreover without loss of generality the considered derivation in $\mathcal{R}$ is ground. Hence we can build only $2^{E \cdot m}$ different terms of size at most m, where E depends only on $\mathcal{F}$. This proves the second inequality. □

Based on Lemmata 26 and 28 we obtain our main theorem.

Theorem 29. *For any finite and terminating TRS $\mathcal{R}$, $\mathsf{dc}_{\mathcal{R}}(n) \leqslant 2^{2^{n \cdot 2^{O(\mathsf{DPc}_{\mathcal{R}}(n))}}}$.*

An order $\succ$ on terms is G-*collapsible* for a TRS $\mathcal{R}$ if $s \to^*_{\mathcal{R} \cup \mathsf{DP}(\mathcal{R})} t$ and $s \succ t$ implies $\mathsf{G}(s, \succ) > \mathsf{G}(t, \succ)$ for a mapping G into $\mathbb{N}$. Let $(\gtrsim, \succ)$ be a reduction pair for $\mathcal{R}$. Then $(\gtrsim, \succ)$ is called *collapsible* if there is a mapping G such that $\succ$ is G-collapsible for $\mathcal{R}$.

Theorem 30. *Let $\mathcal{R}$ be a finite TRS, let $(\gtrsim, \succ)$ be a collapsible reduction pair with $\pi(\mathcal{R}) \subseteq \gtrsim$ and $\pi(\mathsf{DP}(\mathcal{R})) \subseteq \succ$ for some argument filtering π. Assume there exists a class of number-theoretic functions $\mathcal{C}$ closed under elementary functions and for some $f \in \mathcal{C}$, and any term t, $\mathsf{G}(\pi(t^\sharp), \succ) \leqslant f(|t|)$. Then $\mathsf{dc}_{\mathcal{R}} \in \mathcal{C}$.*

Proof. By assumption there exists a mapping G that binds the number of dependency pair steps in any $\pi(\mathcal{R}) \cup \pi(\mathsf{DP}(\mathcal{R}))$-derivation. Thus

$$\mathsf{dl}(\pi(t^\sharp), \to_{\pi(\mathsf{DP}(\mathcal{R}))/\pi(\mathcal{R})}) \leqslant \mathsf{G}(\pi(t^\sharp), \succ) \leqslant f(|t|) \,. \tag{1}$$

Moreover it is easy to see that for any derivation in $\mathcal{R} \cup \mathsf{DP}(\mathcal{R})$, there is a derivation in $\pi(\mathcal{R}) \cup \pi(\mathsf{DP}(\mathcal{R}))$ which contains the same number of dependency pair steps. Hence, we obtain

$$\mathsf{dl}(t^\sharp, \to_{\mathsf{DP}(\mathcal{R})/\mathcal{R}}) \leqslant \mathsf{dl}(\pi(t^\sharp), \to_{\pi(\mathsf{DP}(\mathcal{R}))/\pi(\mathcal{R})}) \,.$$

Combining this with (1) and Theorem 29 we obtain $\mathsf{dc}_\mathcal{R}(n) \leqslant 2^{2^{n \cdot 2^{a \cdot f(n)}}}$. By assumption the complexity class $\mathcal{C}$ is closed under elementary functions. In particular there exists $g \in \mathcal{C}$ such that $\mathsf{dc}_\mathcal{R}(n) \leqslant g(n)$. Thus the theorem follows. □

5 The Lower Bound

By Theorem 29, the derivational complexity of a TRS $\mathcal{R}$ is bounded triple exponentially in its dependency pair complexity. This yields an upper bound. The following TRS establishes a double exponential lower bound.

Example 31. Consider the following TRS $\mathcal{R}_5$, extending the TRS $\mathcal{R}_4$:

$$1\colon\ \mathsf{f}(\mathsf{S}(x), y) \to \mathsf{f}(x, \mathsf{f}(x, y)) \qquad 2\colon\ \mathsf{f}(\mathsf{0}, x) \to \mathsf{c}(x, x)$$

We show that $\mathcal{R}_5$ has linear dependency pair complexity, but admits derivations of double exponential length. Let $F^0_m(x) = x$, $F^{n+1}_m(x) = \mathsf{f}(\mathsf{S}^m(\mathsf{0}), F^n_m(x))$, $C^0(x) = x$, and $C^{n+1}(x) = \mathsf{c}(C^n(x), C^n(x))$. Now, consider the starting term $F^1_n(\mathsf{0})$. As can be easily seen, this term rewrites to $F^{2^n}_0(\mathsf{0})$ in $2^n - 1$ steps using rule 1. Now, we can use rule 2 and an outermost strategy to reach $C^{2^n}(\mathsf{0})$ in $2^{2^n} - 1$ steps, so $\mathsf{dc}_{\mathcal{R}_5}$ is at least double exponential. On the other hand consider $\mathsf{DP}(\mathcal{R}_5)$:

$$3\colon\ \mathsf{f}^\sharp(\mathsf{S}(x), y) \to \mathsf{f}^\sharp(x, \mathsf{f}(x, y)) \qquad 4\colon\ \mathsf{f}^\sharp(\mathsf{S}(x), y) \to \mathsf{f}^\sharp(x, y)$$

We define a (very restricted) polynomial interpretation $\mathcal{A}$ as follows: $\mathsf{f}^\sharp_\mathcal{A}(x, y) = x$, $\mathsf{S}_\mathcal{A}(x) = x+1$, $\mathsf{f}_\mathcal{A}(x, y) = \mathsf{c}_\mathcal{A}(x, y) = \mathsf{0}_\mathcal{A} = 0$, where $\mathcal{R}_5 \subseteq \gtrsim_\mathcal{A}$ and $\mathsf{DP}(\mathcal{R}_5) \subseteq >_\mathcal{A}$, and $(\gtrsim_\mathcal{A}, >_\mathcal{A})$ forms a reduction pair. Thus $\mathsf{DPdc}_{\mathcal{R}_5}$ is at most linear.

Note that from the proof of Theorem 29 one can distill the following three facts, where each of them is responsible for one of the exponentials in the upper bound:

- The number of leaves in a progenitor graph may be exponential in its height.
- The size of a term may be exponential in its depth.
- The number of terms of size n is exponential in n.

Observe that for an optimal example, we would have to utilise all three criteria, while the just given TRS $\mathcal{R}_5$ utilises only the first two criteria. At this point, it seems impossible to enumerate enough terms of exponential depth and double exponential size so that this is possible. Hence, we conjecture that the upper bound given in Theorem 29 can be improved to double exponential.

6 Conclusion

In this paper we have shown that the derivational complexity of a TRS $\mathcal{R}$ is bounded triple exponentially in its dependency pair complexity. Moreover we have presented an example showing that the relationship is at least double exponential. Furthermore, we conjecture that the upper bound can be improved to a double exponential bound.

The basic dependency pair method [5] forms the basis of our investigations. In particular we allow *argument filtering* for the dependency pairs. A similar result can be shown for *dependency graphs*, but we need to replace the triple exponential correspondence by an even faster growing (but still elementary) correspondence, as shown in the extended version of this paper [20]. Future work will concentrate on establishing better bounds for the studied variants, and analysing the derivational complexity induced by further refinements of the dependency pair method.

To summarise the contribution of this paper, we apply Theorem 30 to three well-studied simplification orders: LPO, MPO and KBO. Recall that the derivational complexity induced by LPO or MPO is multiple recursive or primitive recursive, respectively, cf. [11,12]. Clearly these function classes are closed under elementary functions. Hence by Theorem 30 we obtain that the derivational complexity induced by the basic dependency pair method based on LPO (MPO) is multiple recursive (primitive recursive). On the other hand for a TRS $\mathcal{R}$ compatible with KBO we have that $\mathsf{dc}_{\mathcal{R}}$ belongs to $\mathsf{Ack}(\mathsf{O}(n), 0)$, cf. [13]. Thus applying the theorem in the context of KBO yields that the derivational complexity function induced by the dependency pair method based on KBO is majorised by the Ackermann function. Recall that in all three cases the bounds are tight (see [11,12,13]) and using the same examples, we obtain tightness of the here established bounds.

To conclude, we consider a version of the Ackermann function, introduced by Hofbauer [21] in a slightly simpler way, which we denote as $\mathcal{R}_6$.

$$\mathsf{i}(x) \circ (y \circ z) \to \mathsf{f}(x, \mathsf{i}(x)) \circ (\mathsf{i}(\mathsf{i}(y)) \circ z) \qquad \mathsf{i}(x) \to x$$
$$\mathsf{i}(x) \circ (y \circ (z \circ w)) \to \mathsf{f}(x, \mathsf{i}(x)) \circ (z \circ (y \circ w)) \qquad \mathsf{f}(x, y) \to x$$

Note that $\mathcal{R}_6$ is not simply terminating and the derivational complexity of $\mathcal{R}_6$ dominates the Ackermann function. (The latter follows by the same argument as in [21].) However, termination can be shown easily by the basic dependency pair method in conjunction with argument filtering and KBO.

There are nine dependency pairs. For the argument filtering π, we set $\pi(\mathsf{f}) = \pi(\mathsf{f}^\sharp) = \pi(\mathsf{i}^\sharp) = 1$, $\pi(\mathsf{i}) = [1]$, and $\pi(\circ) = \pi(\circ^\sharp) = [1, 2]$. To apply Proposition 2 we use the reduction pair $(\geqslant^\pi_{\mathsf{KBO}}, >^\pi_{\mathsf{KBO}})$ induced by the admissible weight function w with $w_0 = 1$, $w(\circ) = w(\circ^\sharp) = 1$, and $w(\mathsf{i}) = 0$, together with the precedence $\mathsf{i} \succ \circ, \circ^\sharp$. Hence, by Theorem 30 the derivational complexity of $\mathcal{R}_6$ belongs to $\mathsf{Ack}(\mathsf{O}(n), 0)$ and this bound is optimal, compare [13].

Acknowledgements. The second author would like to thank Dieter Hofbauer for his hospitality, and fruitful discussions about a draft of this paper, during the his stay in Kassel.

References

1. Hofbauer, D., Lautemann, C.: Termination proofs and the length of derivations. In: Dershowitz, N. (ed.) RTA 1989. LNCS, vol. 355, pp. 167–177. Springer, Heidelberg (1989)
2. Koprowski, A., Waldmann, J.: Arctic termination ...Below zero. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 202–216. Springer, Heidelberg (2008)
3. Moser, G., Schnabl, A., Waldmann, J.: Complexity analysis of term rewriting based on matrix and context dependent interpretations. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2008. Dagstuhl Seminar Proceedings, vol. 08004, pp. 304–315 (2008)
4. Zantema, H.: Termination of term rewriting by semantic labelling. FI 24(1,2), 89–105 (1995)
5. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. TCS 236(1,2), 133–178 (2000)
6. Moser, G.: Derivational complexity of Knuth Bendix orders revisited. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS, vol. 4246, pp. 75–89. Springer, Heidelberg (2006)
7. Hirokawa, N., Moser, G.: Automated complexity analysis based on the dependency pair method. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 364–379. Springer, Heidelberg (2008)
8. Hirokawa, N., Moser, G.: Complexity, graphs, and the dependency pair method. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS, vol. 5330, pp. 652–666. Springer, Heidelberg (2008)
9. Hirokawa, N., Middeldorp, A.: Automating the dependency pair method. IC 199(1,2), 172–199 (2005)
10. Dershowitz, N.: Termination dependencies. In: Rubio, A. (ed.) WST 2003. Technical Report DSIC-II/15/03, Universidad Politecnica de Valencia, pp. 27–30 (2003)
11. Weiermann, A.: Termination proofs for term rewriting systems with lexicographic path orderings imply multiply recursive derivation lengths. TCS 139(1,2), 355–362 (1995)
12. Hofbauer, D.: Termination proofs by multiset path orderings imply primitive recursive derivation lengths. TCS 105(1), 129–140 (1992)
13. Lepper, I.: Derivation lengths and order types of Knuth-Bendix orders. TCS 269(1,2), 433–450 (2001)
14. Thiemann, R.: The DP Framework for Proving Termination of Term Rewriting. PhD thesis, University of Aachen (2007)
15. Hirokawa, N., Middeldorp, A.: Tyrolean termination tool: Techniques and features. IC 205, 474–511 (2007)
16. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of term rewriting. JAR 40(3), 195–220 (2008)
17. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
18. TeReSe: Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press, Cambridge (2003)
19. Geser, A.: Relative Termination. PhD thesis, Universität Passau (1990)
20. Moser, G., Schnabl, A.: The derivational complexity induced by the dependency pair method. CoRR abs/0904.0570 (2009)
21. Hofbauer, D.: Termination Proofs and Derivation Lengths in Term Rewriting Systems. PhD thesis, Technische Universität Berlin (1992)

Local Termination

Jörg Endrullis[1], Roel de Vrijer[1], and Johannes Waldmann[2]

[1] Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
joerg@few.vu.nl, rdv@cs.vu.nl
[2] Hochschule für Technik, Wirtschaft und Kultur (FH) Leipzig
Fb IMN, PF 30 11 66, D-04251 Leipzig, Germany
waldmann@imn.htwk-leipzig.de

Abstract. The characterization of termination using well-founded monotone algebras has been a milestone on the way to automated termination techniques, of which we have seen an extensive development over the past years. Both the semantic characterization and most known termination methods are concerned with *global* termination, uniformly of all the terms of a term rewriting system (TRS). In this paper we consider *local* termination, of specific sets of terms within a given TRS.

The principal goal of this paper is generalizing the semantic characterization of global termination to local termination. This is made possible by admitting the well-founded monotone algebras to be partial. We show that our results can be applied in the development of techniques for proving local termination. We give several examples, among which a verifiable characterization of the terminating S-terms in CL.

1 Introduction

Along with the growing influence of computers in every part of human society, program verification and termination analysis have become an important branch of computer science. This is the more so since the outbreak of the financial crisis, which has sparked a sharp increase in the demand for efficient and affordable termination tools. An important contribution to the development of automated methods for proving termination has turned out to be the characterization of termination using well-founded monotone algebras.

Both the semantic characterization and most known termination methods are concerned with *global* termination, uniformly of all the terms of a TRS. This is remarkable, as termination is prima facie a property of individual terms. More in general, one may consider the termination problem for an arbitrary set of terms of a TRS. We call this the *local* termination problem.

A typical area where termination techniques are applied is that of program verification. The termination problems naturally arising in program verification are local termination problems: the central interest is termination of a program when started on a valid input. For example in logic programming (e.g. Prolog), local termination has been a central field of research over the past years. Local termination problems of Haskell programs have been considered in [13] and [8].

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 270–284, 2009.
© Springer-Verlag Berlin Heidelberg 2009

In [13], a tableau calculus is devised to show termination of sets of terms of the form $f\, a_1 \ldots a_n$ with the a_i's in normal form. In [8], a transformation from Haskell programs into dependency pair problems [2] is given, which then in turn are solved using methods for global termination.

Surprisingly, for TRSs not much work is known about local termination. We mention the method of match-bounded string rewriting [7], which can be used to prove local termination for sets of strings generated by a regular automaton. Indeed, this method can be viewed as an instance of the semantic framework we develop in this paper.

Local termination is of special interest when dealing with specific classes of terms within a TRS that is known to be non-terminating. Examples of such TRSs are combinatory logic (CL) and encodings of recursive program schemes or Turing machines. The well-known halting problem for Turing machines is a local termination problem. Clearly, this holds for the blank tape halting problem which just asks for termination on the blank tape. On the first glance the uniform halting problem – asking for termination on all inputs – might seem to be global. However, this is a local termination problem as well, since Turing machines are started in a distinguished initial state and admit only one head to work on the tape. In this paper we will use CL and the halting problem for Turing machines to illustrate some of our results (Examples 3.7, 6.7, 6.8, 7.7 and 7.8).

Outline. In Section 3 we generalize the semantic characterization from global termination to local termination based on monotone partial Σ-algebras. This establishes a first, important step towards the development of automatable techniques for proving local termination. In Section 4 we extend this to relative termination, obtaining a characterization using extended monotone partial Σ-algebras.

For global termination it is common practice to stepwise simplify the proof obligation by removing rules. For local termination (the strictly decreasing) rules cannot simply be removed as they influence the set of reachable terms. We need to impose weak conditions on the 'removed' rules, see Section 5.

Having developed the general framework, in the remaining two sections we look for fruitful instances of partial monotone algebras, suitable for automation.

In Section 6 we consider the case that the family of the set of terms for which we want to prove local termination can be described by a partial model. A variant of semantic labeling [17] can then be used to transform the local termination problem into a global termination problem, and the available provers for global termination can be applied. A famous instance is the fragment of combinatory logic based on the single combinator S, this we consider at the end of Section 6.

In Section 7 we combine the partial variant of the quasi-models of [17] with monotone algebras to obtain partial monotone algebras. Partial quasi-models are roughly deterministic tree automata [4] equipped with a relation $\geq$ on the states which guarantees that the language of the automaton is closed under rewriting. Thereby we obtain partial monotone algebras that can successfully be applied for proofs of local termination. Indeed, this method can be automated and, as a matter of fact, we have devised an implementation, which is available via the web page: `http://infinity.few.vu.nl/local/`.

2 Preliminaries

Term rewriting. A *signature* Σ is a non-empty set of symbols, each having a fixed *arity*, given by a map $\sharp : \Sigma \to \mathbb{N}$. Let Σ be a signature and $\mathcal{X}$ a set of variable symbols. The *set* $Ter(\Sigma, \mathcal{X})$ *of terms over* Σ *and* $\mathcal{X}$ is the smallest set satisfying: $\mathcal{X} \subseteq Ter(\Sigma, \mathcal{X})$, and $f(t_1, \ldots, t_n) \in Ter(\Sigma, \mathcal{X})$ if $f \in \Sigma$ with arity n and $\forall i (1 \leq i \leq n) : t_i \in Ter(\Sigma, \mathcal{X})$. We use $x, y, z, \ldots$ to range over variables. Furthermore we use $\equiv$ for syntactical equality of terms. The set of positions $\mathcal{P}os(t) \subseteq \mathbb{N}^*$ of a term $t \in Ter(\Sigma, \mathcal{X})$ is defined as follows: $\mathcal{P}os(f(t_1, \ldots, t_n)) = \{\bot\} \cup \{ip \mid 1 \leq i \leq \sharp(f), p \in \mathcal{P}os(t_i)\}$ and $\mathcal{P}os(x) = \{\bot\}$ for variables $x \in \mathcal{X}$.

A substitution σ is a map $\sigma : \mathcal{X} \to Ter(\Sigma, \mathcal{X})$. For a term $t \in Ter(\Sigma, \mathcal{X})$ we define $t\sigma$ as the result of replacing each $x \in \mathcal{X}$ in t by $\sigma(x)$. Formally, $t\sigma$ is inductively defined by $x\sigma := \sigma(x)$ for variables $x \in \mathcal{X}$ and otherwise $f(t_1, \ldots, t_n)\sigma := f(t_1\sigma, \ldots, t_n\sigma)$. Let $\Box$ be a fresh symbol, $\Box \notin \Sigma \cup \mathcal{X}$. A *context* C is a term from $Ter(\Sigma, \mathcal{X} \cup \{\Box\})$ containing precisely one occurrence of $\Box$. By $C[s]$ we denote the term $C\sigma$ where $\sigma(\Box) = s$ and $\sigma(x) = x$ for all $x \in \mathcal{X}$.

A *term rewriting system (TRS)* R over Σ and $\mathcal{X}$ is a set of pairs $\langle \ell, r \rangle \in Ter(\Sigma, \mathcal{X})$, called *rewrite rules* and written as $\ell \to r$, for which the *left-hand side* ℓ is not a variable, $\ell \notin \mathcal{X}$, and all variables in the *right-hand side* r occur in ℓ as well, $Var(r) \subseteq Var(\ell)$. Let R be a TRS. For terms $s, t \in Ter(\Sigma, \mathcal{X})$ we write $s \to_R t$ (or briefly $s \to t$) if there exists a rule $\ell \to r \in R$, a substitution σ and a context $C \in Ter(\Sigma, \mathcal{X} \cup \{\Box\})$ such that $s \equiv C[\ell\sigma]$ and $t \equiv C[r\sigma]$. The reflexive-transitive closure of $\to$ is denoted as $\twoheadrightarrow$. We call $\to$ the *one-step rewrite relation* induced by R and $\twoheadrightarrow$ the *many-step rewrite* or *reduction relation*. If $t \twoheadrightarrow t'$ then we call t' an *(R)-reduct* of t.

Definition 2.1. Let R be a TRS over Σ and $T \subseteq Ter(\Sigma, \mathcal{X})$ a set of terms. The *family* $\mathcal{F}am_R(T)$ *of* T is the set of subterms of R-reducts of terms $t \in T$.

Partial functions. For partial functions $f : A_1 \times \ldots \times A_n \rightharpoonup A$ and $a_1 \in A_1$, $\ldots$, $a_n \in A_n$ we call $f(a_1, \ldots, a_n)$ *defined* and write $f(a_1, \ldots, a_n)\!\downarrow$ whenever $\langle a_1, \ldots, a_n \rangle$ is in the domain of f. Otherwise $f(a_1, \ldots, a_n)$ is called *undefined* and we write $f(a_1, \ldots, a_n)\!\uparrow$. We use the same terminology and notation for composite expressions involving partial functions. Between such expression we use Kleene equality: $exp_1 \simeq exp_2$ means that either both $exp_1\!\uparrow$ and $exp_2\!\uparrow$, or $exp_1\!\downarrow$ and $exp_2\!\downarrow$ and $exp_1 = exp_2$. Note that an expression can only be defined if all its subexpressions are.

Definition 2.2. Let A be a set and R a relation on A. We define two properties of an n-ary partial function f with respect to R.

(i) f is *closed* if for every $a, b \in A$ we have:

$$f(\ldots, a, \ldots)\!\downarrow \;\;\&\;\; a\, R\, b \;\Rightarrow\; f(\ldots, b, \ldots)\!\downarrow$$

(ii) f is *monotone* if for every $a, b \in A$ we have:

$$f(\ldots, a, \ldots)\!\downarrow \;\;\&\;\; f(\ldots, b, \ldots)\!\downarrow \;\;\&\;\; a\, R\, b \;\Rightarrow\; f(\ldots, a, \ldots)\; R\; f(\ldots, b, \ldots)$$

The functions that we consider will be typically both closed and monotone, which can be rendered briefly as:

$$f(\ldots,a,\ldots)\!\downarrow \;\;\&\;\; a\;R\;b \;\Rightarrow\; f(\ldots,a,\ldots)\;R\;f(\ldots,b,\ldots)$$

By writing something like $exp_1\ R\ exp_2$ we imply that exp_1 and exp_2 are defined.

3 Local Termination

We devise a complete characterization of local termination based on an extension of the monotone algebra approach of [6]. The central idea is the use of monotone partial algebras, that is, the operations of the algebras are allowed to be partial functions. This idea was introduced in [5], where these algebras have been employed to obtain a complete characterization of local infinitary strong normalization. First we give the definition of local termination:

Definition 3.1. Let R be a TRS over Σ, and $T \subseteq Ter(\Sigma,\mathcal{X})$ a set of terms. Then R is called *terminating (or strongly normalizing) on* T, denoted $\mathsf{SN}_R(T)$, if no term $t \in T$ admits an infinite rewrite sequence $t \equiv t_1 \to_R t_2 \to_R \ldots$. We write SN_R for termination on the set of all terms $Ter(\Sigma,\mathcal{X})$.

We give the definition of a partial algebra:

Definition 3.2. A *partial* Σ*-algebra* $\langle A,[\cdot]\rangle$ consists of a non-empty set A and for each n-ary $f \in \Sigma$ a partial function $[f] : A^n \rightharpoonup A$, the *interpretation of* f.

Given a partial Σ-algebra $\mathcal{A} = \langle A,[\cdot]\rangle$ and a (partial) assignment of the variables, $\alpha : \mathcal{X} \rightharpoonup A$, we can give an *interpretation* $[t,\alpha]$ of terms $t \in Ter(\Sigma,\mathcal{X})$, which, however, will not always be defined. So the interpretation is a partial function from terms and partial assignments to A, inductively defined by:

$$[x,\alpha] \simeq \alpha(x)$$
$$[f(t_1,\ldots,t_n),\alpha] \simeq [f]([t_1,\alpha],\ldots,[t_n,\alpha])$$

For ground terms $t \in Ter(\Sigma,\varnothing)$ we write $[t]$ for short.

A set $T \subseteq Ter(\Sigma,\varnothing)$ is called *defined* if for all $t \in T$ we have $[t]\!\downarrow$.

Definition 3.3. A *monotone partial* Σ*-algebra* $\langle A,[\cdot],\succ\rangle$ is a partial Σ-algebra $\langle A,[\cdot]\rangle$ equipped with a binary relation $\succ$ on A such that:

(i) $\succ$ is well-founded,
(ii) for every $f \in \Sigma$ the function $[f]$ is closed and monotone with respect to $\succ$.

Remark 3.4. One could also work with monotone *total* instead of partial algebras, by adding an "undefined" element $\bot$ to the domain. Then defining $\bot$ to be maximal, $\bot \succ a$ for every $a \in A \setminus \{\bot\}$, monotonicity of a function will automatically entail closedness. In order to get full correspondence with our framework of partial algebras, we would in this set-up only consider strict functions.

Definition 3.5. For a monotone partial Σ-algebra $\mathcal{A} = \langle A,[\cdot],\succ\rangle$ we define the TRS R over Σ to be *decreasing (with respect to* $\succ$*)* if for all $\ell \to r \in R$ and every assignment α we have the implication $[\ell,\alpha]\!\downarrow \;\Rightarrow\; [\ell,\alpha] \succ [r,\alpha]$.

The following theorem gives a complete characterization of local termination in terms of monotone partial algebras.

Theorem 3.6. *Let R be a TRS over Σ, and $T \subseteq \mathit{Ter}(\Sigma, \varnothing)$. Then $\mathsf{SN}_R(T)$ holds if and only if there exists a monotone partial Σ-algebra $\mathcal{A} = \langle A, [\cdot], \succ \rangle$ such that T is defined, and R is decreasing.*

Proof. Follows from the more general Theorem 4.5 by Remark 4.2. □

To keep the presentation simple, the theorem characterizes local termination for sets of ground terms $T \subseteq \mathit{Ter}(\Sigma, \varnothing)$ only. Indeed, the theorem can easily be generalized to sets of open terms by, instead of just a monotone partial algebra, additionally requiring a variable assignment α. A set of terms T is then called defined if for that α we have $[t, \alpha]\!\downarrow$ for every $t \in T$.

Example 3.7. We consider the S combinator with the rewrite rule $\mathsf{S}xyz \to xz(yz)$ from combinatory logic, that is, $@(@(@(\mathsf{S}, x), y), z) \to @(@(x, z), @(y, z))$ in first order notation. The S combinator is well-known to be globally non-terminating, however we have local termination on certain sets of terms, for example the set of "flat" S-terms, $T := \{\mathsf{S}^n \mid n \in \mathbb{N},\ n \geq 1\}$, where $\mathsf{S}^1 := \mathsf{S}$ and $\mathsf{S}^{n+1} := @(\mathsf{S}^n, \mathsf{S})$.

We prove strong normalization on T using the monotone partial Σ-algebra $\mathcal{A} = \langle A, [\cdot], \succ \rangle$, where $A := \{\mathsf{s}\} \cup \mathbb{N}$ and the interpretation $[\cdot]$ is given by:

$$[\mathsf{S}] := \mathsf{s} \qquad [@](\mathsf{s}, \mathsf{s}) := 0 \qquad [@](0, n) := n + 1 \qquad [@](n, \mathsf{s}) := 2 \cdot n + 1$$

for all $n \in \mathbb{N}$ and $[@](x, y)\!\uparrow$ for all other cases. Let $\succ$ be the natural order $>$ on $\mathbb{N}$; that is, s is neither source nor target of a $\succ$ step. Then well-foundedness of $\succ$ and monotonicity of $[@]$ are obvious, and T is defined. We have $[\mathsf{S}xyz, \alpha]\!\downarrow$ only if $\alpha(x) = \mathsf{s}$ and $\alpha(z) = \mathsf{s}$; then we obtain:

$$[\mathsf{S}xyz, \alpha] = 3 \succ 1 = [xz(yz), \alpha] \qquad \text{for } \alpha(y) = \mathsf{s}$$
$$[\mathsf{S}xyz, \alpha] = 2 \cdot \alpha(y) + 3 \succ 2 \cdot \alpha(y) + 2 = [xz(yz), \alpha] \qquad \text{for } \alpha(y) \in \mathbb{N}$$

Hence $\mathsf{S}xyz \to xz(yz)$ is decreasing and we conclude termination on T.

4 Local Relative Termination

We define local relative termination.

Definition 4.1. Let R, S be TRSs over Σ, and $T \subseteq \mathit{Ter}(\Sigma, \mathcal{X})$ a set of terms. Then the TRS R is called *terminating (or strongly normalizing) relative to S on T*, denoted $\mathsf{SN}_{R/S}(T)$, if there exists no term $t \in T$ that admits an infinite rewrite sequence $t \equiv t_1 \to_{R \cup S} t_2 \to_{R \cup S} \ldots$ containing an infinite number of $\to_R$ steps. We write $\mathsf{SN}_{R/S}$ for relative termination on all terms $\mathit{Ter}(\Sigma, \mathcal{X})$.

Remark 4.2. Termination of R relative to S on T is equivalent to termination of the relation $\to_R \,/\, \to_S \;:=\; \twoheadrightarrow_S \cdot \to_R \cdot \twoheadrightarrow_S$ on T. Furthermore we have $\mathsf{SN}_R(T)$ if and only if $\mathsf{SN}_{R/\varnothing}(T)$.

Definition 4.3. An *extended monotone partial Σ-algebra* $\langle A, [\cdot], \succ, \sqsupseteq \rangle$ is a monotone partial Σ-algebra $\langle A, [\cdot], \succ \rangle$ with an additional binary relation $\sqsupseteq$ on A for which the following two conditions hold:

(i) $\succ \cdot \sqsupseteq \subseteq \succ$ and $\succ \subseteq \sqsupseteq$ (*compatibility*),
(ii) for every $f \in \Sigma$ the function $[f]$ is closed and monotone with respect to $\sqsupseteq$.

Then a TRS S over Σ is called *weakly decreasing* if for all $\ell \to r \in S$ and every assignment α such that $[\ell, \alpha]\!\downarrow$ we have $[\ell, \alpha] \sqsupseteq [r, \alpha]$.

Lemma 4.4. *Let $\langle A, [\cdot], \succ, \sqsupseteq\rangle$ be an extended monotone partial Σ-algebra and let R and S be TRSs over Σ such that R is decreasing and S weakly decreasing. Furthermore, assume for $s \in \mathit{Ter}(\Sigma, \varnothing)$ that $[s]\!\downarrow$. Then we have the implications: (i) $s \to_R t \;\Rightarrow\; [s] \succ [t]$ and (ii) $s \to_S t \;\Rightarrow\; [s] \sqsupseteq [t]$.*

Proof. The proofs of (i) and (ii) are similar, we just prove (ii). Let $s \to_S t$, that is, we have $\ell \to r \in S$, substitution σ and context C such that $s \equiv C[\ell\sigma]$ and $t \equiv C[r\sigma]$. Since $[s]\!\downarrow$ and $\ell\sigma$ is a subterm also $[\ell\sigma]\!\downarrow$, so $[\ell, \alpha] \sqsupseteq [r, \alpha]$, as S is weakly decreasing. Then using closedness and monotonicity of the interpretations $[f]$ of all function symbols $f \in \Sigma$ we obtain $[s] \sqsupseteq [t]$. □

We give a complete characterization of local relative termination in terms of extended monotone partial algebras.

Theorem 4.5. *Let R and S be TRSs over Σ, and $T \subseteq \mathit{Ter}(\Sigma, \varnothing)$ a set of terms. Then $\mathsf{SN}_{R/S}(T)$ holds if and only if there exists an extended monotone partial Σ-algebra $\mathcal{A} = \langle A, [\cdot], \succ, \sqsupseteq\rangle$ such that the set T is defined, R is decreasing, and S is weakly decreasing.*

Proof. For the 'only if'-part assume that $\mathsf{SN}_{R/S}(T)$ holds. Let $\mathcal{A} = \langle A, [\cdot], \succ, \sqsupseteq\rangle$ where $A := \mathcal{F}am_{R\cup S}(T)$ and the interpretation of a function symbol $f \in \Sigma$ is defined by $[f](t_1, \ldots, t_n) := f(t_1, \ldots, t_n)$ if $f(t_1, \ldots, t_n) \in A$, and $[f](t_1, \ldots, t_n)\!\uparrow$ otherwise. The relations $\sqsupseteq$ and $\succ$ are defined by $\sqsupseteq := \twoheadrightarrow_{R\cup S} \cap (A \times A)$ and $\succ := (\to_R \cdot \twoheadrightarrow_{R\cup S}) \cap (A \times A)$.

We verify that $\mathcal{A}$ is an extended monotone partial Σ-algebra. First note that $\succ \cdot \sqsupseteq \subseteq \succ$ and $\succ \subseteq \sqsupseteq$ hold by definition. Suppose that $\succ$ would not be well-founded. Then there exists $t \in \mathcal{F}am_{R\cup S}(T)$ admitting an infinite $\to_R \cdot \twoheadrightarrow_{R\cup S}$ rewrite sequence, contradicting $\mathsf{SN}_{R/S}(T)$. For $f \in \Sigma$ we show that $[f]$ is closed and monotone with respect to $\succ$ (for $\sqsupseteq$ the reasoning is the same). Consider $s, t \in A$ with $s \succ t$. Whenever $[f](\ldots, s, \ldots)\!\downarrow$ we have also $[f](\ldots, t, \ldots)\!\downarrow$ and $[f](\ldots, s, \ldots) \succ [f](\ldots, t, \ldots)$ as a consequence of the closure of rewriting under contexts. Hence $\mathcal{A}$ is an extended monotone partial Σ-algebra.

The set T is defined, since for every term $s \in T$ we have $[s]\!\downarrow$ by definition. It remains to be proved that R and S are respectively decreasing and weakly decreasing. We only consider R, as the reasoning for S is the same. Let $\ell \to r \in R$ and $\alpha : \mathcal{X} \to A$ such that $[\ell, \alpha]\!\downarrow$. Then $[\ell, \alpha] \equiv \ell\alpha \to_R r\alpha \equiv [r, \alpha]$. Then $[\ell, \alpha] \succ [r, \alpha]$ because both $[\ell, \alpha] \in A$ and $[r, \alpha] \in A$.

For the 'if'-part assume that $\mathcal{A} := \langle A, [\cdot], \succ, \sqsupseteq\rangle$ fulfilling the requirements of the theorem is given. Assume $\mathsf{SN}_{R/S}(T)$ would not hold. Then there exists $t_0 \in T$ which admits an infinite $\to_R \cup \to_S$ rewrite sequence $t_0 \to t_1 \to \ldots$ containing an infinite number of $\to_R$-steps. By Lemma 4.4 this sequence then would give rise to an infinite $\succ \cup \sqsupseteq$ sequence: $[t_0]\ (\succ \cup \sqsupseteq)\ [t_1]\ (\succ \cup \sqsupseteq) \ldots$ containing infinitely many $\succ$ steps. Using compatibility $\succ \cdot \sqsupseteq \subseteq \succ$ we can remove the intermediate $\sqsupseteq$ steps, yielding an infinite $\succ$ sequence, contradicting well-foundedness of $\succ$. □

Example 4.6. We consider a simple example to illustrate the method:

$$R = \{a \to b\} \qquad S = \{b \to b,\ f(b) \to f(a)\} \qquad T = \{a\}$$

Global relative termination $\mathsf{SN}_{R/S}$ does not hold. However on T the rule $a \to b$ is terminating relative to the other rules. We can prove this using the extended monotone partial Σ-algebra $\mathcal{A} = \langle A, [\cdot], \succ, \sqsupseteq \rangle$ where $A = \{0, 1\}$, $1 \succ 0$ and $\sqsupseteq$ is the union of $\succ$ and equality. The interpretations are given by: $[a] = 1$, $[b] = 0$ and $[f](x)\uparrow$ for all $x \in A$. Then T is defined, R is decreasing ($[a] = 1 > 0 = [b]$), and S is weakly decreasing ($[b] = 0 \geq 0 = [b]$ and $[f(b)]\uparrow$). Hence we conclude $\mathsf{SN}_{R/S}(T)$ by an application of Theorem 4.5.

See further Example 7.7 in Section 7 for a non-trivial example.

5 Stepwise Removal of Rules

For termination proofs it is common practice to weaken the proof obligation stepwise by removing rules. The idea is to find interpretations such that a part $R' \subseteq R$ of the rules is decreasing and the remaining rules are weakly decreasing. Then for termination of R it suffices to prove termination of the rules in the complement $R \setminus R'$. We would also like to have this possibility for proofs of local termination. However, for local termination we cannot simply remove (and then forget about) the strictly decreasing rules, as the following example illustrates.

Example 5.1. Consider the set $T = \{a\}$ in the TRS with the following rules:

$$a \to b \qquad b \to b \qquad c \to c$$

We define a monotone partial Σ-algebra $\langle A, [\cdot], \succ \rangle$ by $A = \mathbb{N}$, $[a] = 1$, $[b] = 0$ and $[c]\uparrow$, taking for $\succ$ the natural order $>$ on $\mathbb{N}$. Then $a \to b$ is decreasing since $[a] > [b]$, and for $b \to b$ we have $[b] = [b]$. However, removing the strictly decreasing rule $a \to b$ is not sound, since the resulting TRS is terminating on T.

Let us briefly elaborate on the following theorem which enables us to remove rules stepwise. Assume that the goal is proving that R is terminating relative to S on T, that is, $\mathsf{SN}_{R/S}(T)$. We start with zero knowledge: $\mathsf{SN}_{\varnothing/R \cup S}(T)$. We search for an interpretation that makes a part $R' \subseteq R$ of the rules decreasing and the remaining rules in $R \cup S$ weakly decreasing. Then the rules in R' can only be applied finitely often: $\mathsf{SN}_{R'/((R \setminus R') \cup S)}(T)$. *But how to proceed?* As we have seen above, we cannot simply forget about the rules R', but need to take into account their influence on the family $\mathcal{F}am_{R \cup S}(T)$. A possible and theoretically complete solution would be to require these rules to be weakly decreasing. However, for practical applicability this requirement seems too strict as it imposes heavy restrictions on the termination order. We propose a different approach, which allows the 'removed' rules R' to arbitrarily change, even increase, the interpretation of the rewritten terms, as long as rewriting defined terms yields defined terms again. For this purpose we introduce a relation $\rightsquigarrow$ on A, with respect to which the already removed rules have to be weakly decreasing.

Theorem 5.2. *Let R, R' and U be TRSs over Σ, and $T \subseteq Ter(\Sigma, \varnothing)$ such that $\mathsf{SN}_{U/(R\cup R')}(T)$ holds. Then $\mathsf{SN}_{(U\cup R')/R}(T)$ holds if and only if there exists an extended monotone partial Σ-algebra $\mathcal{A} = \langle A, [\cdot], \succ, \sqsupseteq\rangle$ and a relation $\rightsquigarrow$ on A such that the set T is defined, R' is decreasing, R is weakly decreasing, and:*

- *U is decreasing with respect to $\rightsquigarrow$,*
- *for every $f \in \Sigma$ the function $[f]$ is closed and monotone with respect to $\rightsquigarrow$.*

Proof. Straightforward extension of the proof of Theorem 4.5. The 'only if'-part follows immediately by taking $\rightsquigarrow := \succ$. For the 'if'-part consider an infinite reduction $t_1 \to t_2 \to \dots$ with $t_1 \in T$. Then since U is decreasing with respect to $\rightsquigarrow$ we conclude $\forall i \in \mathbb{N}.\ [t_i]\big\downarrow$ and by $\mathsf{SN}_{U/(R\cup R')}(T)$ we can cut off the prefix of the sequence containing the finitely many U steps. □

For an application of the theorem see Examples 7.7 and 7.8 in Section 7.

6 Via Models from Local to Global Termination

In this section we describe an easy transformation from local to global termination based on an adaptation of semantic labeling [17]. For this purpose we generalise the concept of models from [17] to partial models. Whenever the language T for which we are interested in termination can be described by a partial model, that is, $T = \{t \mid [t]\big\downarrow\}$, then semantic labeling allows for a simple, complete transformation from local to global termination. Here complete means that the original system is locally terminating on T if and only if the transformed, labeled system is globally terminating.

First, we generalise the models from [17] to partial models. A model for a TRS R is a Σ-algebra $\langle A, [\cdot]\rangle$ such that $[\ell, \alpha] = [r, \alpha]$ for every rule $\ell \to r \in R$ and every interpretation $\alpha : Var(\ell) \to A$ of the variables.

Definition 6.1. Let R be a TRS over Σ. A *partial model* $\mathcal{A} := \langle A, [\cdot]\rangle$ *for* R is a partial Σ-algebra $\mathcal{A}$, such that $[\ell, \alpha]\big\downarrow \Rightarrow [\ell, \alpha] = [r, \alpha]$ for every $\ell \to r \in R$ and $\alpha : Var(\ell) \to A$.

Thus the condition $[\ell, \alpha] = [r, \alpha]$ of models is only required if the interpretation of the left-hand side is defined, that is, $[\ell, \alpha]\big\downarrow$. In other words, rewriting may turn an undefined term into a defined term, but not the other way around.

Definition 6.2. Let $\mathcal{A} = \langle A, [\cdot]\rangle$ be a partial model. Then the *language* $\mathcal{L}(\mathcal{A})$ *of* $\mathcal{A}$ is defined as $\mathcal{L}(\mathcal{A}) = \{t \in Ter(\Sigma, \varnothing) \mid [t]\big\downarrow\}$.

We define a variant of semantic labeling where each symbol is labeled by the tuple of the values of its arguments.

Definition 6.3. Let Σ be a signature, and $\mathcal{A} := \langle A, [\cdot]\rangle$ be a partial Σ-algebra. For $t \in Ter(\Sigma, \mathcal{X})$ and $\alpha : Var(t) \to A$ such that $[t, \alpha]\big\downarrow$, the *labeling* $lab_{\mathcal{A}}(t, \alpha)$ *of* t *with respect to* α is defined as follows:

$$lab_{\mathcal{A}}(x, \alpha) := x$$
$$lab_{\mathcal{A}}(f(t_1, \dots, t_n), \alpha) := f^{[t_1,\alpha],\dots,[t_n,\alpha]}(lab_{\mathcal{A}}(t_1, \alpha), \dots, lab_{\mathcal{A}}(t_n, \alpha))\,.$$

over the signature $lab_{\mathcal{A}}(\Sigma) = \{f^{\lambda} \mid f \in \Sigma,\ \lambda \in A^{\sharp(f)} \text{ s.t. } [f](\lambda)\big\downarrow\}$

In order to obtain a complete transformation we need to restrict the models to their core, that is, those elements that are interpretations of ground terms.

Definition 6.4. Let $\mathcal{A} = \langle A, [\cdot] \rangle$ be a partial Σ-algebra. Then the *core* $A_c \subseteq A$ *of* $\mathcal{A}$ is is the smallest set such that $[f](a_1, \ldots, a_n) \in A_c$ whenever $f \in \Sigma$ and $a_1, \ldots, a_n \in A_c$ with $[f](a_1, \ldots, a_n)\big\downarrow$. We say that $\mathcal{A}$ is *core* if $A = \mathcal{A}_c$.

By construction of the core we have $\mathcal{A}_c = \{[t] \mid t \in \mathit{Ter}(\Sigma, \varnothing),\ [t]\big\downarrow\}$. The restriction of a model to its core does not change its language, thus in the sequel we can without loss of generality assume that all models are core.

We have arrived at the transformation from local to global termination. The rules are labeled as known from semantic labeling with the exception that labeled rules are thrown away if the interpretation of their left-hand side is undefined.

Definition 6.5. Let R be a TRS over Σ, and $\mathcal{A} = \langle A, [\cdot] \rangle$ a partial Σ-algebra. We define the *labeling of* R as the TRS $\mathit{lab}_{\mathcal{A}}(R)$ over the signature $\mathit{lab}_{\mathcal{A}}(\Sigma)$ by:

$$\mathit{lab}_{\mathcal{A}}(R) := \{\mathit{lab}_{\mathcal{A}}(\ell, \alpha) \to \mathit{lab}_{\mathcal{A}}(r, \alpha) \mid \ell \to r \in R,\ \alpha : \mathit{Var}(\ell) \to A \text{ s.t. } [\ell, \alpha]\big\downarrow\}\,.$$

A TRS is *collapsing* if it contains rules of the form $\ell \to x$. Such collapsing rules can be eliminated by replacing them with all $\ell\sigma_f \to x\sigma_f$ for every $f \in \Sigma$ where $\sigma_f(x) = f(x_1, \ldots, x_n)$ with $x_1, \ldots, x_n$ pairwise different, fresh variables.

Theorem 6.6. *Let R be a non-collapsing TRS over Σ, and $\mathcal{A} = \langle A, [\cdot] \rangle$ a core partial model for R. Then R is locally terminating on $\mathcal{L}(\mathcal{A})$ if and only if $\mathit{lab}_{\mathcal{A}}(R)$ is globally terminating.*

Proof. We introduce types for $\mathit{lab}_{\mathcal{A}}(R)$ over the sorts A. For every symbol $f^{\lambda} \in \mathit{lab}_{\mathcal{A}}(\Sigma)$ with $\lambda = \langle a_1, \ldots, a_{\sharp(f)} \rangle$ we define f^{λ} to have input sorts $\langle a_1, \ldots, a_n \rangle$ and output sort $[f](a_1, \ldots, a_n)$. Then [12, Proposition 5.5.24] with non-collapsingness of $\mathit{lab}_{\mathcal{A}}(R)$ yields that $\mathit{lab}_{\mathcal{A}}(R)$ is terminating if and only if all well-sorted terms are terminating. Since $\mathcal{A}$ is core there exists a well-sorted ground term for every sort in A. Thus by application of a ground substitution we can assume that all rewrite sequences contain only ground terms, and the set of well-sorted ground terms is exactly the language $\mathcal{L}(\mathcal{A})$ of the model $\mathcal{A}$. □

To apply Theorem 6.6 for proving local termination of R on a set of terms T we have to find a partial model $\mathcal{A}$ for R such that $T \subseteq \mathcal{L}(\mathcal{A})$. Then global termination of $\mathit{lab}_{\mathcal{A}}(R)$ implies local termination of R on T. If moreover we have $\mathcal{F}\mathit{am}(T) = \mathcal{L}(\mathcal{A})$, then the transformation is complete, that is, the converse implication holds as well.

Example 6.7. We revisit Example 3.7 on the S combinator with $T = \{\mathsf{S}^n \mid n \in \mathbb{N}\}$. We choose the partial model $\mathcal{A} = \langle A, [\cdot] \rangle$, where $A = \{0, 1, 2\}$ and the interpretation is defined by: $[\mathsf{S}] = 0$, $[@](0, 0) = 1$, $[@](1, x) = 2$ for all $x \in A$, $[@](2, 0) = 2$, and $\uparrow$ otherwise. Then $T \subseteq \mathcal{L}(\mathcal{A})$ and a short proof even shows that $\mathcal{F}\mathit{am}(T) = \mathcal{L}(\mathcal{A})$. The labeling $\mathit{lab}_{\mathcal{A}}(\{\mathsf{S}xyz \to xz(yz)\})$ is:

$$\begin{aligned}
@^{2,0}(@^{1,0}(@^{0,0}(\mathsf{S}, x), y), z) &\to @^{1,1}(@^{0,0}(x, z), @^{0,0}(y, z)) \\
@^{2,0}(@^{1,1}(@^{0,0}(\mathsf{S}, x), y), z) &\to @^{1,2}(@^{0,0}(x, z), @^{1,0}(y, z)) \\
@^{2,0}(@^{1,2}(@^{0,0}(\mathsf{S}, x), y), z) &\to @^{1,2}(@^{0,0}(x, z), @^{2,0}(y, z))\,.
\end{aligned}$$

The other labeled rules are thrown out as their left-hand side is undefined. Global termination of the transformed system can be shown by the recursive path order.

The Set of Normalizing S-Terms

Here, we consider the fragment CL(S) of CL consisting only of the S combinator. We are aiming at a formal verification of the following proposition:

Proposition 6.8 ([16]). *The set of normalizing* S*-terms is a rational language.*

The next lemma follows easily by considering the Nerode congruence [4].

Lemma 6.9. *For each TRS* R *with* $\mathsf{SN}(t) \Leftrightarrow \mathsf{WN}(t)$ *for every term* $t \in \mathit{Ter}(\Sigma, \varnothing)$*, the minimal complete deterministic bottom-up tree automaton for the language of normalizing terms is a model for* R*. Its reachable part is a partial model.*

In particular this applies for CL(S) since it is orthogonal and non-erasing. The corresponding automaton B is finite (it has 39 states) and can be constructed from the grammar given in [16]. We have formally verified the model property and that the language of B contains all normalizing terms in the proof assistant Coq [1]. For proving that CL(S) is terminating on the language of B we have transformed the local into a global termination problem using Definition 6.5. The resulting TRS contains 1800 rules which are globally terminating, as can be shown using the DP transformation with SCC decomposition [2] together with simple projections and the subterm criterion [9]. We are currently working on the verification of the termination proof in Coq. Information on the ongoing development can be found at `http://www.imn.htwk-leipzig.de/$\sim$waldmann/cl-s/`.

7 Quasi-models for Local Termination

In Sections 3–5 we have devised a characterization of local termination in terms of monotone partial algebras. While this gives the general method, for the purpose of obtaining automatable methods we still lack concrete, fruitful instances of these algebras. For global termination, instances of monotone algebras are well-known. This raises the natural question whether we can transform a given monotone algebra for global termination in such a way to obtain a partial monotone algebra for local termination.

In this section we present one such approach. We extend quasi-models [17] to partial quasi-models and then combine these with (ordinary) monotone algebras. The quasi-models are roughly deterministic tree automata that are closed under rewriting. We then search for such an automaton which accepts the starting language T together with a monotone algebra such that the rewrite rules are decreasing on the language of the automaton. In this way monotone algebras for global termination carry over to local termination and we obtain an automatable method that is applicable for proofs of local termination.

First we give the definition of extended μ-monotone algebras as known from global termination of context-sensitive TRSs, see [11,6]. A mapping $\mu : \Sigma \to \mathbf{2}^{\mathbb{N}}$ is called a *replacement map (for* Σ*)* if for all $f \in \Sigma$ we have $\mu(f) \subseteq \{1, \ldots, \sharp(f)\}$. Let $\langle A, [\cdot] \rangle$ be a Σ-algebra and μ a replacement map. For symbols $f \in \Sigma$ we say

that the interpretation $[f] : A^{\sharp(f)} \to A$ is *μ-monotone* with respect to $\succ$ if for every $a, b \in A$ and $i \in \mu(f)$ with $a \succ b$ we have: $f(\underbrace{\dots}_{i-1}, a, \underbrace{\dots}_{\sharp(f)-i}) \succ f(\dots, b, \dots)$.

Definition 7.1. Let μ be a replacement map for Σ. An *extended μ-monotone Σ-algebra* $\langle A, [\cdot], \succ, \sqsupseteq\rangle$ is a Σ-algebra $\langle A, [\cdot]\rangle$ equipped with two binary relations $\succ$, $\sqsupseteq$ on A for which the following three conditions hold:

(i) $\succ$ is well-founded,
(ii) $\succ \cdot \sqsupseteq \subseteq \succ$ and $\succ \subseteq \sqsupseteq$ (compatibility), and
(iii) for every $f \in \Sigma$ the function $[f]$ is μ-monotone with respect to $\succ$ and $\sqsupseteq$.

We extend quasi-models to partial quasi-models:

Definition 7.2. Let R be a TRS over Σ. A *partial quasi-model* $\mathcal{A} := \langle A, [\cdot], \geq\rangle$ *for* R consists of a Σ-algebra $\langle A, [\cdot]\rangle$, a partial order $\geq$ on A such that:

(i) $[\ell, \alpha]\!\downarrow \Rightarrow [\ell, \alpha] \geq [r, \alpha]$ for every $\ell \to r \in R$ and $\alpha : \mathit{Var}(\ell) \to A$, and
(ii) the function $[f]$ is closed and monotone with respect to $\geq$ for every $f \in \Sigma$.

A quasi-model $\mathcal{A} = \langle A, [\cdot], \geq\rangle$ may contain elements $a \in A$ for which $[t] = a$ implies that t is a normal form. For a given quasi-model the set of these, which we denote by $A_{nf(R)}$, can be computed (see below Definition 7.3). We can exploit this knowledge as follows: if a certain argument of a symbol $f \in \Sigma$ is always a normal form, then the interpretation $[f]$ of f does not need to be monotonic for this argument position. The following definition gives an algorithm for computing the set $A_{nf(R)}$. Elements that are interpretations $[\ell, \alpha]$ of left-hand sides in R cannot belong to this set. Moreover if $a \notin A_{nf(R)}$ and $b = [f](\dots, a, \dots)$ then we conclude $b \notin A_{nf(R)}$. This is formalized as follows:

Definition 7.3. Let R be a TRS over Σ, and $\mathcal{A} = \langle A, [\cdot]\rangle$ a partial Σ-algebra. The *normal forms* $A_{nf(R)}$ *of* $\mathcal{A}$ are the largest set $A_{nf(R)} \subseteq A_c$ such that $[\ell, \alpha] \notin A_{nf(R)}$ for every $\ell \to r \in R$ and every $\alpha : \mathit{Var}(\ell) \to A_c$, and $[f](a_1, \dots, a_n) \notin A_{nf(R)}$ for every $f \in \Sigma$, $a_i \notin A_{nf(R)}$ and $a_1, \dots, a_n \in A_c$.

Then by construction we obtain the following lemma:

Lemma 7.4. *$A_{nf(R)}$ is the set of all elements $a \in A_c$ such that for all terms $t \in \mathit{Ter}(\Sigma, \varnothing)$ it holds: $[t] = a$ implies that t is a normal form with respect to R.*

As mentioned above the interpretations do not need to be monotonic in argument positions which are normal forms. We formalize this by defining a replacement map for the the labeling $\mathit{lab}_{\mathcal{A}}(R)$ of R which does not contain argument positions that are in normal form.

Definition 7.5. Let R be a TRS over Σ, and $\mathcal{A} = \langle A, [\cdot]\rangle$ a partial Σ-algebra. Let the replacement map $\mu^{nf(R)}$ be defined for every symbol $f^\lambda \in \mathit{lab}_{\mathcal{A}}(\Sigma)$ with $\lambda = \langle a_1, \dots, a_{\sharp(f)}\rangle$ as follows: $\mu^{nf(R)}(f^\lambda) = \{1, \dots, \sharp(f)\} \setminus \{i \mid a_i \in A_{nf(R)}\}$.

As an instance of Theorem 5.2 we obtain a method for stepwise rule removal for local termination that is based on a combination of partial quasi-models and extended monotone algebras.

Theorem 7.6. *Let R, R' and U be TRSs over Σ, and $T \subseteq Ter(\Sigma, \varnothing)$ a set of terms such that $\mathsf{SN}_{U/R\cup R'}(T)$ holds. Furthermore let $\mathcal{A} = \langle A, [\cdot], \geq\rangle$ be a partial quasi-model for $R \cup R' \cup U$ with $T \subseteq \mathcal{L}(\mathcal{A})$, and $\mathcal{B} = \langle B, [\cdot]_{\mathcal{B}}, \succ, \sqsupseteq\rangle$ an extended $\mu^{nf(R\cup R')}$-monotone $(lab_{\mathcal{A}}(\Sigma))$-algebra such that:*

(i) the rules in $lab_{\mathcal{A}}(R')$ are decreasing:

$$[\ell, \alpha] \succ [r, \alpha] \text{ for all } \ell \to r \in lab_{\mathcal{A}}(R') \text{ and } \alpha : Var(\ell) \to B\,,$$

(ii) the rules in $lab_{\mathcal{A}}(R)$ are weakly decreasing:

$$[\ell, \alpha] \sqsupseteq [r, \alpha] \text{ for all } \ell \to r \in lab_{\mathcal{A}}(R) \text{ and } \alpha : Var(\ell) \to B\,, \text{ and}$$

(iii) for all $f \in \Sigma$, $\boldsymbol{a}_1\, a\, \boldsymbol{a}_2 \in A^{\sharp(f)}$, $a \geq a' \in A$, and $b_1, \ldots, b_{\sharp(f)} \in B$:

$$[f^{\boldsymbol{a}_1\, a\, \boldsymbol{a}_2}]_{\mathcal{B}}(b_1, \ldots, b_{\sharp(f)}) \sqsupseteq [f^{\boldsymbol{a}_1\, a'\, \boldsymbol{a}_2}]_{\mathcal{B}}(b_1, \ldots, b_{\sharp(f)})\,.$$

Then $\mathsf{SN}_{(U\cup R')/R}(T)$ holds.

Proof (Theorem 7.6). We construct an extended monotone partial Σ-algebra $\mathcal{C} = \langle C, [\cdot]_{\mathcal{C}}, \succ_{\mathcal{C}}, \sqsupseteq_{\mathcal{C}}\rangle$ fulfilling the requirements of Theorem 5.2. Let $C = A \times B$, and define $\langle a_1, b_1\rangle \succ_{\mathcal{C}} \langle a_2, b_2\rangle \iff a_1 \notin A_{nf(R\cup R')}$ & $a_1 \geq a_2$ & $b_1 \succ b_2$ and $\langle a_1, b_1\rangle \sqsupseteq_{\mathcal{C}} \langle a_2, b_2\rangle \iff a_1 \notin A_{nf(R\cup R')}$ & $a_1 \geq a_2$ & $b_1 \sqsupseteq b_2$. Note that the $\mu^{nf(R\cup R')}$-monotonicity is implemented by excluding elements $\langle a_1, b_1\rangle$ with $a_1 \in A_{nf(R\cup R')}$ from being sources of $\succ \cup \sqsupseteq$ steps. Then for every symbol $f \in \Sigma$: $[f]_{\mathcal{C}}(\langle a_1, b_1\rangle, \ldots, \langle a_{\sharp(f)}, b_{\sharp(f)}\rangle) = \langle [f]_{\mathcal{A}}(a_1, \ldots, a_{\sharp(f)}), [f^{a_1,\ldots,a_{\sharp(f)}}]_{\mathcal{B}}(b_1, \ldots, b_{\sharp(f)})\rangle$ if $[f]_{\mathcal{A}}(a_1, \ldots, a_{\sharp(f)})\!\downarrow$, and $\uparrow$ otherwise. Finally, we define the relation $\rightsquigarrow$ on C by $\langle a_1, b_1\rangle \rightsquigarrow \langle a_2, b_2\rangle \iff a_1 \geq a_2$. Now it is straightforward to check that all requirements of Theorem 5.2 are fulfilled, and we conclude $\mathsf{SN}_{(U\cup R')/R}(T)$. □

Let us briefly elaborate on the theorem. As an instance of Theorem 5.2, Theorem 7.6 is applicable for proving local termination as well as local relative termination. We start without knowledge $\mathsf{SN}_{\varnothing/R\cup S}(T)$ and stepwise 'remove' rules, more precisely, we move rules from the right side to the left side of the slash '/'. If we reach the goal $\mathsf{SN}_{R/S}(T)$, then the proof has been successful.

The use of partial quasi-models for $R \cup R' \cup U$ with $T \subseteq \mathcal{L}(\mathcal{A})$ guarantees that the language we consider is closed under rewriting. The set R' is the set of strictly decreasing rules that we are aiming to remove. The $\mu^{nf(R\cup R')}$-monotone $lab_{\mathcal{A}}(\Sigma)$-algebra $\mathcal{B}$ then has the task to make all labeled rules stemming from R' strictly decreasing, and from R weakly decreasing. Then we conclude that $R' \cup U$ is terminating relative to R on T.

Example 7.7 (Klop, see [3], Exercise 7.4.7). Example 3.7 can be generalized to include the combinator K, which has the reduction rule $\mathsf{K}xy \to x$. The initial language of flat S, K-terms is $T = (\mathsf{S}|\mathsf{K})^*$; for example $\mathsf{SSKS} = (((\mathsf{SS})\mathsf{K})\mathsf{S})$. The partial model presented in Example 6.7 can be extended to a partial quasi-model for this generalized example by fixing $[\mathsf{K}] = 0$ and $2 > 0$, $2 > 1$. Note that this is not a model due to $[\mathsf{K}xy, \alpha] = 2 > 0 = [x, \alpha]$ for $\alpha = \lambda z.0$. For the complete proof, employing this quasi model, we refer to: `http://infinity.few.vu.nl/local/`.

The second example illustrates the stepwise rule removal.

Example 7.8. We implement a Turing machine which does the following. Initially it puts two boxes X left and right of its head and afterwards alternately runs left and right between the boxes, each time moving them one position further:

$$\begin{aligned} &wwXwRwwwwwXww \twoheadrightarrow wwXwwwwwwRXww \\ \to\ &wwXwwwwwwLwXw \twoheadrightarrow wwXLwwwwwwwXw \\ \to\ &wXwRwwwwwwwXw \twoheadrightarrow \dots \end{aligned}$$

This is implemented by TRS R consisting of the following rules:

$$\begin{array}{llll} wSw \to XRX & Rw \to wR & RXw \to LwX & RX\Box \to FX\Box \\ & wL \to Lw & wXL \to XwR & \Box XL \to \Box XR \\ & wF \to Fw & wXF \to XwR & \Box XF \to \mathit{finish}\,. \end{array}$$

where all symbols apart form *finish* (which is a constant) are unary, but have been written without parenthesis for the purpose of compactness. Note that the TRS is basically obtained using the standard translation of Turing machines to string rewriting systems, see further [15].

While the Turing machine is terminating on every input, the TRS R fails to be globally terminating. The reason is that R allows for configurations with multiple heads working at the same time on the same tape:

$$\Box XRXwFX\Box \to \Box XLwXFX\Box \to \Box XLXwRX\Box \to^2 \Box XRXwFX\Box \to \dots$$

We will prove that R is locally terminating on all terms containing arbitrary occurrences of the symbols $\Box$, w and at most one occurrence of S, that is, the language given by $T = \{\Box, w\}^* S \{\Box, w\}^* \mathit{finish}$. As the first step we remove the rules $wSw \to XRX$ and $\Box XF \to \mathit{finish}$ using a quasi-model consisting of only one element, accepting all terms, and the interpretation $[w]_{\mathcal{B}}(x) = [\Box]_{\mathcal{B}}(x) = x + 1$ whereas all other symbols are interpreted as $\lambda x.x$.

Second, we use a partial quasi-model $\mathcal{A} = \langle A, [\cdot], \geq \rangle$ where $A = \{0, 1\}$, $0 \geq 0$, $1 \geq 1$, $[\mathit{finish}] = 0$ and the other interpretations are given in the table:

x	$[w](x)$	$[X](x)$	$[R](x)$	$[L](x)$	$[F](x)$	$[\Box](x)$	$[S](x)$
0	0	1	$\uparrow$	$\uparrow$	$\uparrow$	0	0
1	1	0	1	1	1	$\uparrow$	$\uparrow$

As required by the theorem $\mathcal{A}$ is a partial quasi-model for R including the two removed rules $U = \{wSw \to XRX,\ \Box XF \to \mathit{finish}\}$ (without them T would be a normal form). We use this quasi-model together with the extended monotone $\mathit{lab}_{\mathcal{A}}(\Sigma)$-algebra $\mathcal{B} = \langle \mathbb{N}, [\cdot]_{\mathcal{B}}, \succ, \sqsupseteq \rangle$ where $\succ$ and $\sqsupseteq$ are the usual orders $>$ and $\geq$ on $\mathbb{N}$, respectively. The interpretation $[\cdot]_{\mathcal{B}}$ is $[\mathit{finish}]_{\mathcal{B}} = 7$, $[w^0]_{\mathcal{B}}(x) = 2 \cdot x + 1$, $[w^1]_{\mathcal{B}}(x) = 2 \cdot x$, $[X^0]_{\mathcal{B}}(x) = [X^1]_{\mathcal{B}}(x) = x$, $[R^1]_{\mathcal{B}}(x) = 2 \cdot x$, $[L^1]_{\mathcal{B}}(x) = 2 \cdot x + 1$, $[F^1]_{\mathcal{B}}(x) = 2 \cdot x$, $[\Box^0]_{\mathcal{B}}(x) = 2 \cdot x$, and $[S^0]_{\mathcal{B}}(x) = 5 \cdot x + 6$. Then R' consists of the following rules: $RXw \to LwX$, $wL \to Lw$, $wXL \to XwR$, $\Box XL \to \Box XR$, and $wXF \to XwR$. Then $\mathit{lab}_{\mathcal{A}}(R')$ is strictly decreasing with respect to $\mathcal{B}$. For instance consider the rule $RXw \to LwX$. The labeling $R^1X^0w^0 \to L^1w^1X^0$ is

in $lab_{\mathcal{A}}(R')$ and its interpretation in $\mathcal{B}$ is: $R^1X^0w^0(x) = 4 \cdot x + 2 > 4 \cdot x + 1 = L^1w^1X^0(x)$. The labeling $R^0X^1w^1 \to L^0w^0X^1$ is not in $lab_{\mathcal{A}}(R')$ since its left-hand side is undefined with respect to $\mathcal{A}$, thus we can ignore this rule. Analogous it can be verified that all rules in $lab_{\mathcal{A}}(R \setminus R')$ are weakly decreasing in $\mathcal{B}$. Since $>$ is the empty relation on A the third condition of Theorem 7.6 holds trivially.

The three remaining rules $Rw \to wR$, $wF \to Fw$, and $RX\Box \to FX\Box$ are even globally terminating. This corresponds to taking a quasi-model which has only one state and accepts all terms together with the corresponding termination order which proves global termination. Hence we have proven $\mathsf{SN}_R(T)$ by three consecutive applications of Theorem 7.6.

Finally, we give a theorem that allows to remove rules and forget about them. We need to be sure that these rules do not influence the family. This is guaranteed if all terms in the family are normal forms with respect to these rules.

Theorem 7.9. *Let R, R' and S be TRSs over Σ, and $T \subseteq Ter(\Sigma, \varnothing)$. Moreover let $\mathcal{A} = \langle A, [\cdot], \geq \rangle$ be a partial quasi-model for $R \cup R' \cup S$ with $T \subseteq \mathcal{L}(\mathcal{A})$ such that for all rules $\ell \to r \in R'$ and $\alpha : Var(\ell) \to A$ we have $[\ell, \alpha]\uparrow$ (the left-hand side is undefined). Then $\mathsf{SN}_{R/S}(T)$ implies $\mathsf{SN}_{R \cup R'/S}(T)$.*

Proof. From $\mathcal{F}am_{R \cup R' \cup S}(T) \subseteq \mathcal{L}(\mathcal{A})$ together with $[\ell, \alpha]\uparrow$ for all $\ell \to r \in R'$ and α it follows that the rules in R' are not reachable. All terms in $\mathcal{F}am(T)$ are normal forms with respect to R'. Hence we can ignore these rules. □

Example 7.10. Consider the TRS R consisting of the following four rules:

$$f(\mathsf{s}(\mathsf{s}(x))) \to f(o(x)) \quad o(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{s}(o(x))) \quad o(0) \to 0 \quad o(\mathsf{s}(0)) \to \mathsf{s}(\mathsf{s}(0)))$$

The TRS is not terminating: $f(\mathsf{s}(\mathsf{s}(0)))) \to f(o(\mathsf{s}(0))) \to f(\mathsf{s}(\mathsf{s}(0)))) \to \ldots$. However, the function f is terminating when applied to an even number, that is, the language $T = \{f(\mathsf{s}^{2 \cdot n}(0)) \mid n \in \mathbb{N}\}$. We choose $\mathcal{A} = \langle \{0, 1\}, [\cdot], \geq \rangle$ where $[0] = 0$, $[\mathsf{s}](0) = 1$, $[\mathsf{s}](1) = 0$, $[o](0) = 0$, $[o](1)\uparrow$, $[f](0) = 0$ and $[f](1)\uparrow$. Then $\mathcal{A}$ is a partial quasi-model with $T \subseteq \mathcal{L}(\mathcal{A})$. We have $[o(\mathsf{s}(0)), \alpha]\uparrow$ (for all α), thus the rule $o(\mathsf{s}(0)) \to \mathsf{s}(\mathsf{s}(\mathsf{s}(0)))$ is never applicable and can be removed.

8 Conclusion

We mention some directions of ongoing and future work.

We intend to generalize the characterization of local termination to context-sensitive rewriting [11], using μ-monotonic, partial Σ-algebras; and also to top termination, using weakly extended, monotone, partial Σ-algebras [2,6].

It seems quite feasible to formalize and verify the characterization of the set of terminating S-terms in CL from Example 6.8 in the proof assistant Coq.

Methods using transformations from certain properties, like liveness properties [10] or outermost termination [14], to termination usually give rise to local termination problems. That is, termination is of interest only for those terms which are in the image of the transformation. For example, we noted that the transformation in [14] gives rise to a language which can be described by a partial model. Then it suffices to show completeness of the transformation to local

termination, and employing Theorem 6.6 we obtain a complete transformation to global termination for free.

References

1. The Coq Proof Assistant, `http://coq.inria.fr/`
2. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoretical Computer Science 236, 133–178 (2000)
3. Barendregt, H.P.: The Lambda Calculus, Its Syntax and Semantics. Studies in Logic and the Foundation of Mathematics, vol. 103. Elsevier, Amsterdam (1984)
4. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2007), `http://www.grappa.univ-lille3.fr/tata`
5. Endrullis, J., Grabmayer, C., Hendriks, D., Klop, J.W., de Vrijer, R.C.: Proving Infinitary Normalization. In: Berardi, S., Damiani, F., de'Liguoro, U. (eds.) TYPES 2008. LNCS, vol. 5497, pp. 64–82. Springer, Heidelberg (2009)
6. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of term rewriting. Journal of Automated Reasoning (2008)
7. Geser, A., Hofbauer, D., Waldmann, J.: Match-bounded string rewriting systems. Appl. Algebra Eng., Commun. Comput. 15(3), 149–171 (2004)
8. Giesl, J., Swiderski, S., Thiemann, R., Schneider-Kamp, P.: Automated termination analysis for Haskell: From term rewriting to programming languages. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 297–312. Springer, Heidelberg (2006)
9. Hirokawa, N., Middeldorp, A.: Dependency pairs revisited. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 249–268. Springer, Heidelberg (2004)
10. Koprowski, A.: Termination of Rewriting and Its Certification. PhD thesis, Eindhoven University of Technology (2008)
11. Lucas, S.: Context-Sensitive Computations in Functional and Functional Logic Programs. Journal of Functional and Logic Programming 1998(1) (1998)
12. Ohlebusch, E.: Advanced Topics in Term Rewriting. Springer, New York (2002)
13. Panitz, S.E., Schmidt-Schauß, M.: TEA: Automatically proving termination of programs in a non-strict higher-order functional language. In: Van Hentenryck, P. (ed.) SAS 1997. LNCS, vol. 1302. Springer, Heidelberg (1997)
14. Raffelsieper, M., Zantema, H.: A transformational approach to prove outermost termination automatically. Technical report, RISC-Linz Report Series (2008)
15. Terese: Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press, Cambridge (2003)
16. Waldmann, J.: The combinator S. Information and Computation 159, 2–21 (2000)
17. Zantema, H.: Termination of term rewriting by semantic labelling. Fundamenta Informaticae 24, 89–105 (1995)

VMTL–A Modular Termination Laboratory

Felix Schernhammer* and Bernhard Gramlich

TU Wien, Austria
{felixs,gramlich}@logic.at

Abstract. The automated analysis of termination of term rewriting systems (TRSs) has drawn a lot of attention in the scientific community during the last decades and many different methods and approaches have been developed for this purpose. We present VMTL (Vienna Modular Termination Laboratory), a tool implementing some of the most recent and powerful algorithms for termination analysis of TRSs, while providing an open interface that allows users to easily plug in new algorithms in a modular fashion according to the widely adopted dependency pair framework. Apart from modular extensibility, VMTL focuses on analyzing the termination behaviour of conditional term rewriting systems (CTRSs). Using one of the latest transformational techniques, the resulting restricted termination problems (for unconditional context-sensitive TRSs) are processed with dedicated algorithms.

1 Introduction and Overview

During the last decade, remarkable progress has been made in the field of termination analysis of term rewriting systems. Despite termination being an undecidable property of TRSs, increasingly sophisticated methods have been developed to prove it for given systems. From these efforts several tools have emerged that are capable of proving termination (semi) automatically (a list of currently available tools can be obtained from the official website of the termination competition [18].)

The currently most powerful tools implement the dependency pair framework of [8] based on the idea of dependency pair analysis of [3]. This approach has at least two advantages. First, it seems to be the most powerful one, which is indicated by the latest results of a yearly termination competition ([18]). Second, the pure dependency pair framework is strictly modular, which means that concrete (correct) methods to prove termination within this framework can be combined, added and removed arbitrarily, without affecting the correctness of the tool. Thus, it is easy to extend tools implementing this framework by new methods (called *dependency pair processors* in the terminology of [8]).

VMTL (currently available in Version 1.1, cf. `http://www.logic.at/vmtl/`) is a new termination tool that implements the dependency pair framework and focuses on openness, modularity and extensibility. It is easily extensible by new

* The author has been supported by the Austrian Academy of Sciences under grant 22.361.

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 285–294, 2009.
© Springer-Verlag Berlin Heidelberg 2009

dependency pair processors, while providing the main technical infrastructure of termination tools such as thread and processor scheduling, timeout handling, input parsing, output formatting etc. In addition, VMTL contains implementations of several well-known methods of proving termination within the dependency pair framework. These include

- a dependency graph processor used for decomposing dependency pair problems into strongly connected components of an estimated dependency graph (the estimation described in [1, Section 4.1] is used),
- a reduction pair processor using recursive path orderings with status based on [1, Theorem 21],
- a reduction pair processor based on polynomial interpretations featuring linear and simple mixed polynomials, with coefficients from $\mathbb{N}$ and constants from $\mathbb{Z}$,
- forward and backward narrowing as well as instantiation processors (see below resp. [17, Section 6] for more details), and
- a size-change principle processor based on results from [19].

The set of implemented processors (in particular the inclusion of the size change principle processor) was chosen to optimize the power of VMTL with respect to proving termination of conditional term rewriting systems. Moreover, all of these processors are "context-restriction aware" in VMTL which means that they are sound for both context-sensitive and standard termination problems.

The two reduction pair processors are implemented via reduction to satisfiability problems and utilizing external SAT solvers, as it is state-of-the-art at the time of writing. Benchmarks of VMTL on the set of TRSs used in the latest termination competition can be found below.

Another focus during the development of VMTL was applicability to (and suitability for) conditional term rewriting systems (CTRSs). Termination of such CTRSs is usually verified by transforming them into ordinary TRSs and deriving termination of the CTRSs from termination of the transformed TRSs. VMTL provides a public interface that allows users to plug in such transformations. Moreover, it includes a recent one ([17]) that transforms CTRSs into context-sensitive (unconditional) TRSs.

VMTL is also capable of proving termination of context-sensitive term rewriting systems (CSRSs). For this task the refined context-sensitive dependency pair approach of [1] (cf. also [2]) is used. Using context-sensitive dependency pairs, and their property that they coincide with context-free (i.e., standard) dependency pairs for context-free (i.e., ordinary) term rewriting systems, allows VMTL to treat every TRS as CSRS and analogously treat every standard dependency pair problem as context-sensitive one. Still, the use of dedicated DP processors for each kind of problem is possible in VMTL (see Section 4.1 below).

2 Preliminaries

We assume familiarity with the basic concepts and notations of term rewriting and context-sensitive rewriting (cf. e.g. [4,5,12]). As we will briefly discuss

VMTL's approach to prove operational termination of deterministic conditional term rewriting systems (DCTRSs) in Section 5, we introduce some basic notions regarding conditional term rewriting.

Conditional rewrite systems consist of conditional rules $l \to r \Leftarrow c$, with c of the form $s_1 \to^* t_1, \ldots, s_n \to^* t_n$ such that l is not a variable and $Var(r) \subseteq Var(l) \cup Var(c)$. In the following we are concerned with *deterministic* 3-CTRSs (DCTRSs), which have the additional property that for each conditional rule $l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n$ it holds that $Var(s_i) \subseteq Var(l) \cup \bigcup_{j=1}^{i-1} Var(t_j)$. Deterministic 3-CTRSs are arguably the most general class of CTRSs for which termination analysis makes sense, as in non-deterministic CTRSs arbitrary instantiations of extra variables usually entail that the system is not (effectively [16] / operationally [13]) terminating.

The conditional rewrite relation induced by a CTRS $\mathcal{R}$ is inductively defined as follows: $R_0 = \emptyset$, $R_{j+1} = \{\sigma l \to \sigma r \mid l \to r \Leftarrow s_1 \to^* t_1, \ldots, s_n \to^* t_n \in \mathcal{R}, \sigma s_i \to^*_{R_j} \sigma t_i \text{ for all } 1 \leq i \leq n\}$, and $\to_{\mathcal{R}} = \bigcup_{j \geq 0} \to_{R_j}$. In contrast to ordinary term rewriting systems well-foundedness of the induced rewrite relation is not an adequate notion of termination for CTRSs. The proof-theoretic notion of *operational termination* has turned out to be adequate to guarantee finiteness of derivations in CTRSs (cf. [13,17] for details).

Given a CSRS $\mathcal{R} = (\Sigma, R)$ with replacement map μ, the relation of context-sensitive narrowing (written $\rightsquigarrow^{\mu}_{\mathcal{R}}$) is defined as $t \rightsquigarrow^{\mu}_{\mathcal{R}} s$ if there is a replacing non-variable position p in t such that $t|_p$ and l unify (where $l \to r \in R$ and where we assume that t and l do not share any variables) with mgu θ and $s = \theta(t[r]_p)$. We say that s is a *one-step, context-sensitive narrowing* of t.

2.1 The Context-Sensitive Dependency Pair Framework

The dependency pair framework of [8] resp. [1] basically reduces termination problems to problems of proving finiteness of so-called *dependency pair problems*.

We call a triple $(P, \mathcal{R}, \mu)$, where P and $\mathcal{R}$ are TRSs and μ is a replacement map for the combined signatures of P and $\mathcal{R}$, a *(context-sensitive) dependency pair problem* (CS-DP-problem).[1] Such a problem is finite if P terminates *relative* to $(\mathcal{R}, \mu)$ where $\to_P$ steps may only occur as root steps and $\to_{\mathcal{R},\mu}$ steps may only occur strictly below the root.

Within the dependency pair framework these problems are analyzed by so-called *dependency pair processors*.

A *CS-dependency pair processor* is a function $Proc$ that takes as input a CS-DP-problem and returns either a set of CS-dependency pair problems or "`no`". We call a CS-DP-processor *sound* if finiteness of all CS-DP-problems in $Proc(d)$ implies finiteness of the input CS-DP-problem d. A CS-DP-processor is *complete* if for all CS-DP-problems d, d is infinite whenever $Proc(d)$ is "`no`" or $Proc(d)$ contains an infinite CS-DP-problem.

For further details regarding the dependency pair framework we refer to [8].

[1] Initially, P consists of the (context-sensitive) dependency pairs of $(\mathcal{R}, \mu)$ or $\mathcal{R}$, respectively, cf. [1,3].

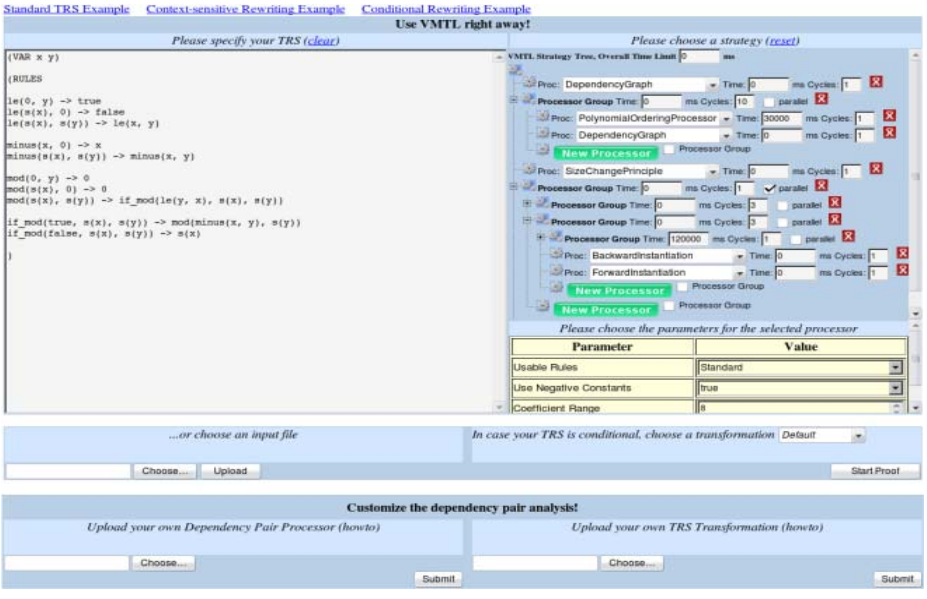


Fig. 1. Snapshot of the VMTL web interface after specifying a particular proof strategy

3 User Interface

VMTL provides a command line interface for batch execution and a web interface that eases configuration and extension. Figure 1 shows a screenshot of the web interface, after specifying a particular proof strategy. It contains fields for entering a TRS or alternatively uploading a file. VMTL exclusively accepts the input format specified for the termination competition. On the right-hand side the user can define the strategy, i.e., the order in which processors are applied, together with time constraints to be satisfied. At the bottom there are two fields allowing the user to upload new customized dependency pair processors and transformations, respectively.

3.1 User Defined Strategies

Since dependency pair processors often modify and simplify dependency pair problems, it is crucial to apply them in a reasonable order. Moreover, some processors are more time consuming than others. Thus, in order to achieve the goal of proving termination as quickly as possible, it is important to have a good strategy for processor application. VMTL allows the user to fully configure this strategy. It provides the following degrees of freedom in its specification.

- Dependency pair processors can be arbitrarily ordered.
- Time limits can be imposed on dependency pair processors.

- Dependency pair processors can be executed repeatedly (which can be helpful for instance for *narrowing* processors).
- Dependency pair processors can be hierarchically grouped to an arbitrary depth. Each group can be given a time limit. In case of contradicting time limits the shortest one is used.
- Execution of (groups of) dependency pair processors can be parallelized.

The semantics of parallelism in VMTL is that if one of the parallel branches finishes the termination (or non-termination) proof, the whole execution stops immediately and the proof is presented to the user. Otherwise, if all parallel branches fail to prove termination, the termination proof continues according to the strategy with the dependency pair problems derived *before* the start of the parallel execution or with any of the problems derived in the parallel paths, depending on which of these problems is the "most simple" one. This is determined by a configurable measure function on dependency pair problems. By default, this function compares the size of the problems to determine the "most simple" one.

Graphically, a strategy is represented in VMTL as a tree, where each node can either be a dependency pair processor, or a "Group node". Group nodes are used to build groups of processors or other groups, cf. Figure 1.

VMTL provides a default strategy that turned out to be a reasonable compromise between power and efficiency, which allows users to test systems for termination without manually specifying a proof strategy. This standard strategy was also used in the benchmarks below.

4 VMTL API

VMTL provides a public java programming interface that allows a user to easily build extensions. There are three basic functionalities that can be extended.

- New dependency pair processors can be added.
- New transformations, from (conditional/context-sensitive) TRSs to (context-sensitive) TRSs, can be added.
- New plug-ins for output formatting can be added.

4.1 Adding New Dependency Pair Processors

VMTL provides two interfaces *DPProcessor* and *ContextSensitiveDpProcessor* from which one has to be implemented by the user depending on whether the processor takes context-restrictions into account or not. In case DPProcessor is implemented, VMTL will make sure that the processor is not applied to context-sensitive dependency pair problems (even if the processor occurs in the strategy). Note that each context-sensitive DP processor is trivially a context-free one (as context-free DP problems can be seen as special cases of context-sensitive ones), thus ContextSensitiveDpProcessor is a "subinterface" (in an object oriented sense) of DPProcessor. See e.g. [10, Example 3] for a justification that

context-free dependency pair processors (for instance using *usable rules*) may in general not be applied to context-sensitive dependency pair problems.

Technically, a user-defined processor is given a dependency pair problem (that has possibly been processed by other processors before) and a set of (processor) parameters from VMTL and is required to return a set of derived dependency pair problems. In VMTL, the datastructure of a dependency pair problem consists of two sets of rewrite rules and a replacement map according to the definition in Section 2.1. Additionally, in VMTL it contains a subsignature (cf. [17, Section 6]) and several additional flags (in particular one for position-based strategies such as *innermost rewriting* and one for completeness indicating whether infinity of the dependency pair problem implies non-termination of the initial rewrite system).

4.2 Adding New Transformations

Adding transformations to VMTL can be accomplished by implementing the interface *TrsToTrsTransformation*. Transformations in VMTL are not restricted to ones that transform conditional systems. The interface can be used to perform arbitrary preprocessing steps, such as semantic labelling etc. (this is the reason why the interface is not called *CtrsToTrsTransformation*). However, in case termination of a conditional rewrite system is to be proved, a transformation is mandatory. If none is specified by the user, VMTL will use the context-sensitive unraveling transformation of [6,15,17].

4.3 Customizing Output Formatting

The proof information, that is accumulated by VMTL, and the used dependency pair processors are represented in a simple native markup language, providing basic structuring and formatting tags. From this intermediate representation the actual (human or machine) readable output is created by so-called *OutputWriter* objects. Out of the box, VMTL only supports HTML output. However, the user may extend VMTL by additional OutputWriters through the *OutputWriter* interface and can thereby provide additional support for proof certification (see e.g. [11]).

5 Termination of CTRSs

In ([17]) it was shown that when verifying (operational) termination of a CTRS by transformation, it is sufficient to prove termination of the transformed system only on a restricted set of terms. This idea is incorporated in VMTL by extending (context-sensitive) dependency pair problems as defined in [8] by an additional component representing a sub-signature, which identifies the set of terms for which termination is to be shown.

In VMTL the following context-sensitive version ([17]) of an unraveling transformation ([14,16]) is included.

Definition 1. *(context-sensitive unraveling of DCTRSs) Let $\mathcal{R}$ be a DCTRS ($\mathcal{R} = (\Sigma, R)$). For every 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 in the following way:*

$$\begin{aligned} l &\to U_1^\alpha(s_1, Var(l)) \\ U_1^\alpha(t_1, Var(l)) &\to U_2^\alpha(s_2, Var(l), \mathcal{E}Var(t_1)) \\ &\vdots \\ U_n^\alpha(t_n, Var(l), \mathcal{E}Var(t_1), \ldots, \mathcal{E}Var(t_{n-1})) &\to r \end{aligned}$$

Here $Var(s)$ denotes an arbitrary but fixed sequence of the set of variables of the term s. Let $\mathcal{E}Var(t_i)$ be $Var(t_i) \setminus (Var(l) \cup \bigcup_{j=1}^{i-1} Var(t_j))$. Abusing notation, by $\mathcal{E}Var(t_i)$ we mean an arbitrary but fixed sequence of the variables in the set $\mathcal{E}Var(t_i)$. Any unconditional rule of $\mathcal{R}$ is transformed into itself. The transformed system $U_{cs}(\mathcal{R}) = (U(\Sigma), U(R))$ is obtained by transforming each rule of $\mathcal{R}$ where $U(\Sigma)$ is Σ extended by all new function symbols. The replacement map $\mu_{U(\Sigma)}$ is given by $\mu_{U(\Sigma)}(f) = \{1, \ldots, ar(f)\}$ for all $f \in \Sigma$ and $\mu_{U(\Sigma)}(f) = \{1\}$ for all $f \in U(\Sigma) \setminus \Sigma$.

For this transformation it turns out that termination of the transformed TRS *on original terms*, i.e., on terms over the original signature of the conditional system, is sufficient (and indeed equivalent) to derive (operational) termination of the DCTRS (cf. [17, Theorems 4, 5 and Corollary 3]). In particular, this means that potential infinite reduction sequences issuing from (non-original) terms built over the extended signature of the transformed system may be ignored.

This generalizes previous results of [6] and [15]. In [6] general termination of $U_{cs}(\mathcal{R})$ was used as a sufficient condition for operational termination of $\mathcal{R}$ (however, there the transformation dealt with a wider class of conditional systems). In [15] it was shown that derivations (from original terms to original terms) in the transformed system correspond to derivations in the original conditional systems if the reduction in the transformed system follows a certain reduction strategy. We refer to [17] for a more thorough discussion of our above characterization result for operational termination of DCTRSs.

VMTL takes advantage of these facts by adapting its narrowing processors. Inside the dependency pair framework a narrowing processor basically exploits the fact that in a dependency pair chain the reduction sequence between two dependency pairs must be non-empty if the right-hand side of the first pair does not unify with the left-hand side of the second one. The (forward) narrowing processor then anticipates the possible first steps of this reduction sequence that affect the right-hand side of the first dependency pair $l \to r$ and replaces the latter by new pairs $\theta_1 l \to r_1, \ldots, \theta_n l \to r_n$ where $\{r_1, \ldots, r_n\}$ is the set of (context-sensitive) one-step narrowings of r and θ_i are the involved mgu's.

According to [17, Definitions 10–13, Theorems 7–10], only a subset of these narrowings need to be considered. Although heuristics are needed to approximate the relevant terms here, this helps to counter the explosion of the number of dependency pairs one often has to deal with when using narrowing processors.

6 Implementation Details and Benchmarks

VMTL (version 1.1) is entirely written in Java. The core module (i.e., without user interface) contains approximately 11.500 lines of code. MiniSat ([7]) is used as SAT solver.

Currently, there is no dedicated category for conditional TRSs in the termination problem database. Still, a few conditional TRSs are contained in it. Table 1 shows the benchmarks of VMTL and AProVE (Version 1.2)[2] on these examples, extended by further examples taken from [9,14,16]. Numbers in parentheses of the "Successful Proofs" column mean successful disproofs of termination. All experiments had a time limit of 60 seconds and were performed on an Intel E8500 (3.16 GHz) with 4GB of RAM under Ubuntu Hardy Heron (32 Bit) using java 6 Build 13.

Table 1. Benchmarks on conditional TRSs

Tool	Successful Proofs	Number of Systems
VMTL	19(3)	24
AProVE	14(2)	24

Table 2. Benchmarks on standard TRSs from the TPDB

Tool	Successful Proofs	Number of Systems
VMTL	629(106)	1391
AProVE	1226(231)	1391
TTT2	970(178)	1391
Jambox	810(60)	1391

Table 3. Benchmarks on context-sensitive TRSs from the TPDB

Tool	Successful Proofs	Number of Systems
VMTL	72(2)	109
AProVE	94(0)	109
MU-TERM	82(0)	109

For several examples used in the table, proving termination heavily depends on the methods described in [17, Section 6].

Tables 2 and 3 show the performance of VMTL on the set of standard TRSs and the set of context-sensitive ones, respectively, from the TPDB. More details regarding all benchmarks, including execution times and actual termination proofs can be found on the VMTL homepage[3].

[2] Newer versions of AProVE as well as other termination tools do not seem to support proper deterministic conditional rewrite systems (i.e., DCTRS with extra variables).

[3] `http://www.logic.at/vmtl/`

Note that apart from the restricted set of proof methods available in VMTL, the inferior performance for standard and context-sensitive TRSs is due to the strict modularity. Strategies in VMTL cannot be history-aware, which for instance prevents using methods like *safe narrowing* from [8]. In addition, the proof strategy cannot be adapted to certain classes of TRSs such as applicative ones.

7 Conclusion, Related and Future Work

We introduced the termination tool VMTL that provides an easily extensible implementation of the dependency pair framework. In particular, interfaces for dependency pair processors, preprocessing steps (e.g. transformations) and output preparation are available. VMTL is supposed to support researchers in the field of termination analysis, who do not have direct access to one of the existing termination tools. Using VMTL they may relatively easily try out and evaluate their ideas without having to build their own implementation from scratch. The system also comprises implementations of some standard termination analysis methods, that should make it even easier to obtain useful benchmarks. Moreover, VMTL provides implementations of state-of-the-art methods for the termination analysis of conditional and context-sensitive rewrite systems.

Regarding conditional rewrite systems, VMTL provides the infrastructure to follow the transformational approach of [17], that reduces the problem of proving (operational) termination of CTRSs to the problem of proving termination of a (transformed context-sensitive) TRS *on a restricted set of terms*. A simple method to exploit these results is realized in terms of two generalized narrowing processors, that enable VMTL to prove termination on original terms even if general termination does not hold.

Many of the currently developed termination tools follow the dependency pair framework. Although VMTL is not yet competitive for standard and context-sensitive TRSs, we see at least two points that distinguish VMTL from other existing tools:

- VMTL provides open and easily accessible interfaces for extensions.
- VMTL provides dedicated methods for the termination analysis of conditional rewrite systems that go beyond reduction to standard termination problems.

Future extensions of VMTL may provide means for an even more fine-grained guidance of proof attempts through a more powerful strategy language (adding for instance non-determinism). Moreover, an extension to new classes of rewrite systems and termination problems (e.g. higher-order rewrite systems and innermost/outermost or relative termination) and programming languages (e.g. Haskell) as input is desirable.

References

1. Alarcón, B., Emmes, F., Fuhs, C., Giesl, J., Gutiérrez, R., Lucas, S., Schneider-Kamp, P., Thiemann, R.: Improving context-sensitive dependency pairs. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS, vol. 5330, pp. 636–651. Springer, Heidelberg (2008)
2. Alarcón, B., Gutiérrez, R., Lucas, S.: Context-sensitive dependency pairs. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 297–308. Springer, Heidelberg (2006)
3. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoretical Computer Science 236(1–2), 133–178 (2000)
4. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press, New York (1998)
5. Bezem, M., Klop, J.W., de Vrijer, R. (eds.): Term rewriting systems. Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press, Cambridge (2003)
6. Durán, F., Lucas, S., Meseguer, J., Marché, C., Urbain, X.: Proving operational termination of membership equational programs. Higher-Order and Symbolic Computation 21, 59–88 (2008)
7. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
8. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. Journal of Automated Reasoning 37(3), 155–203 (2006)
9. Gmeiner, K., Gramlich, B.: Transformations of conditional rewrite systems revisited. In: Corradini, A., Montanari, U. (eds.) Recent Trends in Algebraic Development Techniques (WADT 2008) – Selected Papers. LNCS. Springer, Heidelberg (to appear, 2009)
10. Gutiérrez, R., Lucas, S., Urbain, X.: Usable rules for context-sensitive rewrite systems. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 126–141. Springer, Heidelberg (2008)
11. Koprowski, A.: Termination of rewriting and its certification. PhD thesis, Eindhoven University of Technology (2008)
12. Lucas, S.: Context-sensitive computations in functional and functional logic programs. Journal of Functional and Logic Programming 1998(1) (January 1998)
13. Lucas, S., Marché, C., Meseguer, J.: Operational termination of conditional term rewriting systems. Inf. Process. Lett. 95(4), 446–453 (2005)
14. Marchiori, M.: Unravelings and ultra-properties. In: Hanus, M., Rodríguez-Artalejo, M. (eds.) ALP 1996. LNCS, vol. 1139, pp. 107–121. Springer, Heidelberg (1996)
15. Nishida, N., Sakai, M., Sakabe, T.: Partial inversion of constructor term rewriting systems. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 264–278. Springer, Heidelberg (2005)
16. Ohlebusch, E.: Advanced topics in term rewriting. Springer, London (2002)
17. Schernhammer, F., Gramlich, B.: Characterizing and proving operational termination of deterministic conditional term rewriting systems. Technical Report E1852-2009-01, TU Wien (March 2009), `http://www.logic.at/vmtl/`
18. The termination competition, `http://termination-portal.org/wiki/Termination_Competition`
19. Thiemann, R., Giesl, J.: The size-change principle and dependency pairs for termination of term rewriting. Applicable Algebra in Engineering, Communication and Computing 16(4), 229–270 (2005)

Tyrolean Termination Tool 2*

Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp

Institute of Computer Science
University of Innsbruck
6020 Innsbruck, Austria
ttt2@informatik.uibk.ac.at

Abstract. This paper describes the second edition of the *Tyrolean Termination Tool*—a fully automatic termination analyzer for first-order term rewrite systems. The main features of this tool are its (non-)termination proving power, its speed, its flexibility due to a strategy language, and the fact that the source code of the whole project is freely available. The clean design together with a stand-alone OCaml library for term rewriting, make it a perfect starting point for other tools concerned with rewriting as well as experimental implementations of new termination methods.

Keywords: term rewriting, termination, automation.

1 Introduction

Termination of term rewrite systems (TRSs) is an undecidable property. Nevertheless a vast number of methods have been developed to determine termination, many of which are suitable for implementation. This paper summarizes the main design issues, features, and successes of the Tyrolean Termination Tool 2 (T$_{\text{T}}$T$_2$ for short), the completely redesigned successor of the award winning[1] Tyrolean Termination Tool (T$_{\text{T}}$T) [10]. T$_{\text{T}}$T$_2$ is a tool for automatically proving termination of TRSs, based on the dependency pair framework [7, 8, 10, 22]. It incorporates several novel methods like increasing interpretations, a modular match-bound technique, uncurrying, and outermost loops, which are not (yet) available in other termination provers. It produces readable output and has a simple web interface. Precompiled binaries, sources and documentation of T$_{\text{T}}$T$_2$ are available at

`http://cl-informatik.uibk.ac.at/software/ttt2/`

In contrast to its predecessor, T$_{\text{T}}$T$_2$ is open source; published under terms of the GNU Lesser General Public License. This work refers to version 1.0 of the tool.

The remainder of the paper is organized as follows. In the next section we describe how T$_{\text{T}}$T$_2$ can be used from the command line and sketch its web interface.

* This research is supported by FWF (Austrian Science Fund) project P18763.
[1] Best paper award, RTA 2003.

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 295–304, 2009.
© Springer-Verlag Berlin Heidelberg 2009

Section 3 explains the strategy language of T$_{\text{T}}$T$_2$, which gives the user full control over the implemented termination methods. Some of the available termination techniques are listed in Section 4. In Section 5 we discuss the performance of our tool in light of the latest issue of the termination competition before addressing ongoing and future work in Section 6. We conclude in Section 7.

2 Design

The tool is written in OCaml[2] and consists of about 30,000 lines of code. Approximately 13% are dedicated to provide some general useful functions and data structures. Another 24% are used to implement the rewriting library which deals with terms and rules. The biggest fragment—about 49%—is used to implement termination methods and the strategy language. The rest (about 14%) is concerned with input and output. Since our tool provides several techniques that modify a termination problem by transforming it into different problem domains, T$_{\text{T}}$T$_2$ interfaces the SAT solver MiniSat [2] and the SMT solver Yices [1]. For interfacing C code the third party contribution CamlIDL[3] is needed. The use of monads to implement the strategy language and several other parts of the tool, allow a clean and abstract treatment of the internal prover state in a purely functional way. Additionally, monads facilitate changes (like the integration of a new termination method).

Besides the actual termination prover, we provide the following libraries:

- `util` extends the functionality of several modules from the standard OCaml library. Furthermore modules for graph manipulation, advanced process and timer handling, as well as monads are included.
- `parsec` is an OCaml port of the Haskell parsec[4] library, i.e., the implementation of a functional combinator parser library.
- `rewriting` provides types and functions dealing with terms, substitutions, contexts, TRSs, etc. The functionality is not only aimed at termination, e.g., the computation of overlaps and normal forms is also supported.
- `logic` provides an OCaml interface that abstracts over the two constraint solvers MiniSat and Yices. To this end arithmetical formulas are encoded in an intermediate datatype. When solving the constraints the user specifies the back-end. In the case of MiniSat, additional information (how many bits are used to represent numbers and intermediate results) can be provided. Afterwards the propositional formula is transformed into conjunctive normal form by a satisfiability-preserving transformation [19]. Yices, on the other hand, does neither require the number of bits as a parameter nor the transformation due to built-in support for *linear* arithmetic and formulas not in conjunctive normal form.
- `processors` collects the numerous (non-)termination methods.
- `ttt2` contains the strategy language and connects the preceding libraries.

[2] `http://caml.inria.fr/`
[3] `http://caml.inria.fr/pub/old_caml_site/camlidl/`
[4] `http://legacy.cs.uu.nl/daan/parsec.html`

2.1 Command Line Interface

In order to run TTT2 from the command line, the user can either download the source code from the TTT2 web page and install it following the installation guidelines or alternatively download the binary of the latest version of TTT2. After a successful installation, TTT2 can be started via the command

```
./ttt2 [options] <file> [timeout]
```

where `[options]` denotes a list of command line options, `<file>` specifies the name of the file containing the TRS of which termination should be proved, and `[timeout]`—a floating point number—defines the time limit for proving termination of the given TRS. The TRS must adhere to the termination problem database format.[5] The timeout is optional. To get a complete list of the command line options of TTT2 either read the documentation provided on the tool's homepage or execute the command `./ttt2 --help`.

2.2 Web Interface

The web interface of TTT2 allows the user to play around with some termination methods and an automatic strategy. The design is intentionally simple to abstract from the challenging task to provide a fast and powerful automatic strategy.

3 The Strategy Language

As mentioned in the introduction, TTT2 is designed according to the dependency pair framework which ensures that all methods are implemented in a modular way. In order to combine these methods in a flexible manner, TTT2 provides a *strategy language*. In the following the most important constructs of this language are explained. For further information please consult the online documentation.

3.1 Syntax

The operators provided by the strategy language can be divided into three classes: *combinators*, *iterators*, and *specifiers*. Combinators are used to combine two strategies whereas iterators are used to repeat a given strategy a designated number of times. In contrast, specifiers are used to control the behavior of strategies. The most common combinators are the infixes '`;`', '`|`', and '`||`'. The most common iterators are the postfixes '`?`', '`+`', and '`*`'. The most common specifier is '`[`f`]`' (also written postfix), where f denotes some floating point number. In order to obtain a well-formed strategy s, these operators have to be combined according to the grammar

$$s ::= m \mid \texttt{(}s\texttt{)} \mid s\texttt{;}s \mid s\texttt{|}s \mid s\texttt{||}s \mid s\texttt{?} \mid s\texttt{+} \mid s\texttt{*} \mid s\texttt{[}f\texttt{]}$$

[5] `http://www.lri.fr/~marche/tpdb/format.html`

where m denotes any available method of T$_{\text{T}}$T$_2$ (possibly followed by some flags). In order to avoid unnecessary parentheses, the following precedence is used: `?`, `+`, `*`, `[`f`]` $>$ `;` $>$ `|`, `||`.

3.2 Semantics

In the remainder of this section we use the notion *termination problem* to denote a TRS, a dependency pair problem (DP problem), or a relative termination problem. We call a termination problem *terminating* if the underlying TRS (DP problem, relative termination problem) is terminating (finite, relative terminating). A strategy works on a termination problem. Whenever T$_{\text{T}}$T$_2$ executes a strategy, internally, a so called proof object is constructed which represents the actual termination proof. Depending on the shape of the resulting proof object after applying a strategy s, we say that s *succeeded* or s *failed*.

This should not be confused with the possible answers of the prover: `YES`, `NO`, and `MAYBE`. Here `YES` means that termination could be proved, `NO` indicates a successful non-termination proof, and `MAYBE` refers to the case when termination could neither be proved nor disproved. On success of a strategy s it depends on the internal proof object whether the final answer is `YES` or `NO`. On failure, the answer always is `MAYBE`. Based on the two possibilities success or failure, the semantics of the strategy operators is as follows.

The combinator '`;`' denotes sequential composition. Given two strategies s and s' together with a termination problem P, ``$s$`;`$s'$`` first tries to apply s to P. If this fails, then also ``$s$`;`$s'$`` fails, otherwise s' is applied to the resulting termination problem, i.e., the strategy ``$s$`;`$s'$`` fails, whenever one of s and s' fails. The combinator '`|`' denotes choice. Different from sequential composition, the choice `|` succeeds whenever at least one of s or s' succeeds. More precisely, given the strategy `|`, T$_{\text{T}}$T$_2$ first tries to apply s to P. If this succeeds, its result is the result of `|`, otherwise s' is applied to P. The combinator '`||`' is quite similar to the choice combinator and denotes parallel execution. That means given the strategy `||`, T$_{\text{T}}$T$_2$ runs s and s' in parallel on the termination problem P. As soon as at least one of s and s' succeeds, the resulting termination problem is returned. This can be seen as a kind of non-deterministic choice, since on simultaneous success of both s and s', it is more or less arbitrary whose result is taken.

Example 1. Consider the following strategy:

```
dp;edg;sccs;(bounds -dp || (matrix -wm | kbo -af))
```

In order to prove termination of a TRS $\mathcal{R}$ using this strategy, T$_{\text{T}}$T$_2$ first computes the dependency pairs $\mathcal{P}$ of $\mathcal{R}$ using the `dp` processor (thereby transforming the initially supplied TRS into a DP problem). After that the estimated dependency graph and the strongly connected components of the DP problem $(\mathcal{P}, \mathcal{R})$ are computed, resulting in a set of DP problems $\{(\mathcal{P}_1, \mathcal{R}), \ldots, (\mathcal{P}_n, \mathcal{R})\}$. Finally, to conclude that the DP problem $(\mathcal{P}, \mathcal{R})$ is finite and hence that the TRS $\mathcal{R}$ is terminating, T$_{\text{T}}$T$_2$ tries to prove finiteness of each DP problem $(\mathcal{P}_i, \mathcal{R})$ with

$1 \leqslant i \leqslant n$ by running the match-bound technique and a combination of the matrix method and the Knuth-Bendix order in parallel. Note that the substrategy `matrix -wm | kbo -af` first applies the matrix method to a given termination problem and on failure, applies the Knuth-Bendix order. Here the flag `-wm` indicates that weakly monotone interpretations are used and the flag `-af` specifies that argument filterings should be considered when computing the ordering.

Next we describe the iterators '`?`', '`+`', and '`*`'. The strategy s`?` tries to apply the strategy s to a termination problem P. On success its result is returned, otherwise P is returned unmodified, i.e., s`?` applies s once or not at all to P and always succeeds. The iterators '`+`' and '`*`' are used to apply s recursively to P until P cannot be modified any more. The difference between '`+`' and '`*`' is that s`*` always succeeds whereas s`+` only succeeds if it can prove or disprove termination of P. In other words, s`*` is used to *simplify* problems, since it applies s until no further progress can be achieved and then returns the latest problem. In contrast '`+`' requires the proof attempt to be completed.

Example 2. We extend the strategy of the previous example by adding the iterator '`+`' and two new methods:

```
uncurry?;poly -ib 2 -ob 4*;
 dp;edg;(sccs;(bounds -dp || (matrix -wm | kbo -af)))+
```

To prove termination of a TRS $\mathcal{R}$, T$_{\text{T}}$T$_2$ performs the following steps. At first uncurrying is applied. Since this method works only for applicative TRSs, the iterator '`?`' is added in order to avoid that the whole strategy fails if $\mathcal{R}$ is not an applicative system. After that polynomial interpretations with two input bits (coefficients) and four output bits (intermediate results) are used to simplify the given TRS. (Restricting the values for intermediate computations results in efficiency gains.) The iterator '`*`' ensures that a maximal number of rewrite rules is removed by applying the method as often as possible. Finally, after the computation of the dependency pairs and the estimated dependency graph, T$_{\text{T}}$T$_2$ tries to prove finiteness of the given DP problems, by applying the strategy `sccs;(bounds -dp || (matrix -wm | kbo -af))` recursively.

At last we explain the specifier '`[`f`]`' which denotes timed execution. Given a strategy s and a timeout f, s`[`f`]` tries to modify a given termination problem P for at most f seconds. If s does not succeed or fail within f seconds (wall clock time), s`[`f`]` fails. Otherwise s`[`f`]` succeeds and returns the termination problem that remains after applying s to P.

Example 3. To ensure that the strategy of the previous example is executed for at most 5 seconds we add the specifier '`[5]`'. In addition we limit the time spend by the match-bound technique to 1 second.

```
(uncurry?;poly -ib 2 -ob 4*;
 dp;edg;(sccs;(bounds -dp[1] || (matrix -wm | kbo -af)))+)[5]
```

Using this strategy, T$_{\text{T}}$T$_2$ has at most 5 seconds to prove termination of a given TRS $\mathcal{R}$ and in each iteration 1 second is available to simplify termination problems using the match-bound technique. If the 5 seconds expire, the execution is aborted immediately.

3.3 Specification and Configuration

In order to call TTT2 with a certain strategy, the flag `--strategy` (or alternatively the short form `-s`) has to be set. For convenience it is possible to call TTT2 without specifying any strategy. In this case a predefined strategy is used (for details execute `./ttt2 --help`). Note that the user is responsible for ensuring soundness of the strategy, e.g., applying the processors in correct order.

Example 4. To call TTT2 with the strategy of Example 3, the following command is used: `./ttt2 -s '(uncurry?;poly -ib 2 -ob 4*; ...)[5]' <file>`. Alternatively, one could also remove the outermost time limit of the strategy and pass it as an argument to TTT2. In that case the command looks as follows: `./ttt2 -s '(uncurry?;poly -ib 2 -ob 4*; ...)' <file> 5`.

Since strategies can get quite complex (e.g., the strategy used in the November 2008 termination competition consists of about 100 lines), TTT2 provides the opportunity to specify a configuration file. This allows to abbreviate and connect different strategies. By convention strategy abbreviations are written in capital letters. To tell TTT2 which configuration file should be used, the flag `--conf` (or the short form `-c`) followed by the file name has to be set.

Example 5. Consider the strategy of Example 3. In order to call TTT2 with this strategy we write a configuration file `ttt2.conf` containing the following lines:

```
[Abbreviations]
PRE = uncurry?;poly -ib 2 -ob 4*
PARALLEL = (bounds -dp[1] || (matrix -wm | kbo -af))
AUTO = (PRE;dp;edg;(sccs;PARALLEL)+)[5]
```

It is important to note that abbreviations are not implicitly surrounded by parentheses since this allows more freedom in abbreviating expressions. To tell TTT2 that the strategy `AUTO` of the configuration file `ttt2.conf` should be used the following flags have to be specified: `./ttt2 -c ttt2.conf -s AUTO <file>`.

4 A Selection of Implemented Techniques

In this section some characteristic methods of TTT2 are presented.

Bounds. TTT2 provides the match-bound technique [5] which uses tree automata techniques to prove termination of a TRS on a particular language (in general the set of all ground terms). To increase the applicability of the match-bound technique, TTT2 was the first tool that incorporated it—in a fully modular way—into the dependency pair framework [14]. Moreover, match-bounds can be used to prove complexity results. It is well-known that match-bounds imply linear derivational complexity for non-duplicating systems [5].

KBO. TTT2 employs the most sophisticated implementations of the Knuth-Bendix ordering. The proof obligations are formulated as a propositional formula (set of pseudo boolean constraints, linear arithmetic constraints) [24] and then solved by MiniSat (MiniSat+,Yices).

Loops. Besides techniques from [18], for string rewrite systems (SRSs) loops are searched with the help of SAT solving. After fixing parameters such as the maximal length of words and the maximal length of the non-terminating sequence, loops are encoded in propositional logic [25]. Additionally, based on the approach from [18], a novel idea [23] allows to check whether a given loop also is an outermost loop, i.e., a loop under the outermost reduction strategy.

Matrices. T$_{\mathsf{T}}$T$_2$ implements matrix [12, 3] and arctic [13] interpretations. Our tool uses higher dimensions than competitors which sometimes results in very short and elegant termination proofs. A direct termination proof by arctic matrices yields linear derivational complexity and direct matrix interpretations (of triangular shape) [15] give polynomial upper bounds.

Polynomials. Apart from polynomial interpretations over different carriers (natural numbers, integers, rationals), additional power is achieved by allowing approximations of minimum and maximum operations [4]. Furthermore, techniques from [26] allow an increase in the constant part of the interpretation for some rules.

Root-Labeling. T$_{\mathsf{T}}$T$_2$ was the first tool that incorporated root-labeling within the dependency pair framework [21]. As a result, in 2007 it was the first automated tool that could prove termination of an SRS with non-primitive recursive derivation length (`Zantema/z090`). Since root-labeling preserves derivational complexity it is a viable transformation for proofs of complexity.

Uncurrying. T$_{\mathsf{T}}$T$_2$ incorporates uncurrying for non-proper systems [9] similar to its predecessor T$_{\mathsf{T}}$T. Furthermore it integrates the method within the dependency pair framework [11]. This makes it a very strong tool on the subclass of applicative systems. Due to the fact that reductions in the uncurried system are strictly longer compared to the original system, upper bounds for complexity considerations are not affected by this transformation.

5 T$_{\mathsf{T}}$T$_2$ in Action

It goes without saying that T$_{\mathsf{T}}$T$_2$ is not the only tool for proving termination of rewrite systems. Since 2004 some of these automated termination analyzers compete against each other in regular competitions.[6] In the following paragraphs we compare our tool with some of the other systems that participated in the latest editions of the international termination competition.[7] T$_{\mathsf{T}}$T$_2$ participated in three categories with the aim to show its flexibility and speed.

[6] `http://termination-portal.org/wiki/Termination_Competition`
[7] `http://termcomp.uibk.ac.at/`

SRS Standard. TTT2 won this category in front of AProVE [6] and Jambox.[8] The main reason was that we used matrix and arctic interpretations of higher dimensions than AProVE and Jambox. Proofs were found for the systems `Trafo/un02`, `Trafo/un15`, `Trafo/un17`, and the randomly generated `Waldmann07b/size-12-alpha-3-num-469` which could not be handled by any other tool (also not in previous competitions).

TRS Standard. TTT2 finished second behind AProVE but in front of Jambox. The main emphasis was put on speed. TTT2 could (dis)prove termination of 970 TRSs out of 1391 TRSs in less than ten minutes. That means, it could handle 79% of the systems AProVE could answer but in just 10% of the time.

TRS Outermost. TTT2 could solve twice as much systems as each of the three competitors. While all other tools employed transformations that allowed to use methods designed for full termination, TTT2 integrated a direct approach for finding loops under a specific strategy [23].

TTT2 is not only successful on its own. Two derivatives of TTT2 were involved in other categories of the competition, namely GaT[9] (which was developed by the authors) and TCT [16] (an independent tool, built on top of the basic components of TTT2). TCT was the first tool dedicated to proving complexity certificates. The aim of GaT was just to show how helpful it is to start from the basis of a well-designed termination prover; the additional implementation effort took a single day. GaT won both categories (*Derivational Complexity – Full Rewriting* and *Derivational Complexity – Innermost Rewriting*) in which it participated. Another tool that builds on TTT2 is MKBTT [20], which implements multi-completion using external termination provers. Experimental results revealed that due to thousands of calls to the external prover, a fast one is preferable over a powerful one. This observation inspired our configuration of TTT2 for the *TRS Standard* category in the November 2008 competition where we used less than 10% of the allowed time, to show how many problems could be solved in a strongly limited amount of time.

6 Future Work

Two main goals for the near future are: the improvement of the output produced by TTT2 and the formalization and certification of (non-)termination methods. Concerning the output we plan to transform the internal proof objects into XML. Afterwards it should be possible to convert this XML format into either human readable output or a proof format suitable for automatic certification. For the second goal a parallel project addresses the formalization of rewriting (IsaFoR, Isabelle Formalization of Rewriting) in the theorem prover Isabelle/HOL [17].

[8] `http://joerg.endrullis.de/`

[9] `http://cl-informatik.uibk.ac.at/software/cat/`

This formalization deals with rewriting in general and (non-)termination based on the dependency pair framework in particular. In order to be usable for automated certification of proofs generated by a termination tool, we employ Isabelle's code-generation facilities to export verified Haskell code. This results in the program CeTA[10] (Certification of Termination Analysis), capable of certifying (non-)termination proofs.

7 Conclusion

In this paper we described the termination prover T$_{\text{T}}$T$_2$, the successor of the well-known Tyrolean Termination Tool. We presented its strategy language, some of its characteristic methods, and we compared T$_{\text{T}}$T$_2$ with other termination provers to show its flexibility and versatility. We conclude the paper by listing what we believe to be the main attractions of T$_{\text{T}}$T$_2$:

- it is open source,
- it provides a strategy language which allows to configure it for all possible applications,
- it benefits from multi-core architecture due to support for parallelism,
- it is one of the fastest and most powerful termination provers, and
- it provides stand-alone libraries for parsing, rewriting, and logic.

Acknowledgments. We thank Nao Hirokawa for providing the sources of T$_{\text{T}}$T and Sarah Winkler for writing a first interface for MiniSat.

References

1. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006)
2. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
3. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of rewrite systems. Journal of Automated Reasoning 40(2-3), 195–220 (2008)
4. Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: Maximal termination. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 110–125. Springer, Heidelberg (2008)
5. Geser, A., Hofbauer, D., Waldmann, J., Zantema, H.: On tree automata that certify termination of left-linear term rewriting systems. Information and Computation 205(4), 512–534 (2007)
6. Giesl, J., Schneider-Kamp, P., Thiemann, R.: AProVE 1.2: Automatic termination proofs in the dependency pair framework. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 281–286. Springer, Heidelberg (2006)

[10] `http://cl-informatik.uibk.ac.at/software/ceta/`

7. Giesl, J., Thiemann, R., Schneider-Kamp, P.: The dependency pair framework: Combining techniques for automated termination proofs. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS (LNAI), vol. 3452, pp. 301–331. Springer, Heidelberg (2005)
8. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. Journal of Automated Reasoning 37(3), 155–203 (2006)
9. Hirokawa, N., Middeldorp, A.: Uncurrying for termination. In: Kesner, D., van Raamsdonk, F., Stehr, M.O. (eds.) HOR 2006, pp. 19–24 (2006)
10. Hirokawa, N., Middeldorp, A.: Tyrolean Termination Tool: Techniques and features. Information and Computation 205(4), 474–511 (2007)
11. Hirokawa, N., Middeldorp, A., Zankl, H.: Uncurrying for termination. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 667–681. Springer, Heidelberg (2008)
12. Hofbauer, D., Waldmann, J.: Termination of string rewriting with matrix interpretations. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 328–342. Springer, Heidelberg (2006)
13. Koprowski, A., Waldmann, J.: Arctic termination ... below zero. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 202–216. Springer, Heidelberg (2008)
14. Korp, M., Middeldorp, A.: Match-bounds revisited. Information and Computation (2009), doi:10.1016/j.ic.2009.02.010
15. Moser, G., Schnabl, A., Waldmann, J.: Complexity analysis of term rewriting based on matrix and context dependent interpretations. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2008. DROPS, vol. 1762, pp. 304–315. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Dagstuhl (2008)
16. Moser, G., Schnabl, A.: Proving quadratic derivational complexities using context dependent interpretations. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 276–290. Springer, Heidelberg (2008)
17. Nipkow, T., Paulson, L.C., Wenzel, M.T.: Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002)
18. Payet, É.: Loop detection in term rewriting using the eliminating unfoldings. Theoretical Computer Science 403(2-3), 307–327 (2008)
19. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. Journal of Symbolic Computation 2(3), 293–304 (1986)
20. Sato, H., Winkler, S., Kurihara, M., Middeldorp, A.: Multi-completion with termination tools (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 306–312. Springer, Heidelberg (2008)
21. Sternagel, C., Middeldorp, A.: Root-labeling. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 336–350. Springer, Heidelberg (2008)
22. Thiemann, R.: The DP Framework for Proving Termination of Term Rewriting. PhD thesis, RWTH Aachen, Available as technical report AIB-2007-17 (2007)
23. Thiemann, R., Sternagel, C.: Loops under strategies. In: Treinen, R. (ed.) RTA 2009. LNCS, vol. 5595, pp. 17–31. Springer, Heidelberg (2009)
24. Zankl, H., Hirokawa, N., Middeldorp, A.: KBO orientability. Journal of Automated Reasoning (2009), doi:10.1007/s10817-009-9131-z
25. Zankl, H., Middeldorp, A.: Nontermination of string rewriting using SAT. In: Hofbauer, D., Serebrenik, A. (eds.) WST 2007, pp. 56–59 (2007)
26. Zankl, H., Middeldorp, A.: Increasing interpretations. Annals of Mathematics and Artificial Intelligence (to appear, 2009)

From Outermost to Context-Sensitive Rewriting

Jörg Endrullis and Dimitri Hendriks

Vrije Universiteit Amsterdam
{joerg,diem}@few.vu.nl

Abstract. We define a transformation from term rewriting systems (TRSs) to context-sensitive TRSs in such a way that termination of the target system implies outermost termination of the original system. For the class of left-linear TRSs the transformation is complete. Thereby state-of-the-art termination methods and automated termination provers for context-sensitive rewriting become available for proving termination of outermost rewriting. The translation has been implemented in Jambox, making it the most successful tool in the category of outermost rewriting of the last edition of the annual termination competition.

1 Introduction

Termination is a key aspect of program correctness, and therefore a widely studied subject in term rewriting and program verification. While termination is undecidable in general, various automated techniques have been developed for proving termination. One of the most powerful techniques is the method of dependency pairs [2]. Recently [1], this method has been generalized to context-sensitive TRSs, thereby significantly extending the class of context-sensitive TRSs for which termination can be shown automatically. Context-sensitive rewriting [6] is a restriction on term rewriting where rewriting in some fixed arguments of function symbols is disallowed. It offers a flexible paradigm to analyze properties of rewrite strategies, in particular of (lazy) evaluation strategies employed by functional programming languages.

In this paper context-sensitive rewriting is the target formalism for a transformational approach to the problem of outermost termination, that is, termination with respect to outermost rewriting. Outermost rewriting is a rewriting strategy where a redex may be contracted as long as it is not a proper subterm of another redex occurrence. The main reason for studying outermost termination is its practical relevancy: lazy functional programming languages like Miranda, Haskell or Clean, are based on outermost rewriting as an evaluation strategy, and in implementations of rewrite logic such as Maude and CafeOBJ, outermost rewriting can be specified. Consider the TRS R_0 consisting of the following rules:

$$a \to f(a) \qquad\qquad f(f(x)) \to b \qquad\qquad (R_0)$$

Clearly, this system is not terminating as witnessed by the infinite rewrite sequence $a \to f(a) \to f(f(a)) \to f(f(f(a))) \to \ldots$, but it is outermost terminating. Indeed, the third step in the infinite sequence is not an outermost step, since

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 305–319, 2009.
© Springer-Verlag Berlin Heidelberg 2009

the contraction takes place inside another redex. The only (maximal) outermost rewrite sequence the term a admits is $a \to f(a) \to f(f(a)) \to b$.

Our contribution is a transformation of arbitrary TRSs into context-sensitive TRSs (μTRSs) in such a way that rewriting in the μTRS corresponds to outermost rewriting in the original TRS. As a result advanced termination techniques for μTRSs become applicable for proving outermost termination, and automated termination provers for μTRSs can directly (without modification, only preprocessing) be used for proving outermost termination. Our transformation is complete for the class of quasi left-linear TRSs (a generalized form of left-linear TRSs, see [8]), that is, termination of the resulting μTRS is equivalent to outermost termination of the original system.

The transformation is comprised of a variant of semantic labeling [12]. In semantic labeling the function symbols in a term are labeled by the interpretation of their arguments (or a label depending on these values) according to some given semantics. We employ semantic labeling in order to mark symbols at redex positions, and we obtain a μTRS by defining a replacement map that disallows rewriting inside arguments of marked symbols.

We illustrate our use of semantic labeling by the TRS R_0 from the first page. We choose the algebra $\mathcal{A}_0 = \langle \{0,1\}, [\cdot] \rangle$ where the interpretation indicates the presence of the symbol f, thus $[a] = [b] = 0$, and $[f](x) = 1$ for $x \in \{0,1\}$. Then we write $f^\star$ if the value of its argument is 1, and just f if the value is 0. The symbol a is always marked, and b never is. If f is marked it corresponds to a redex position with respect to the rule $f(f(x)) \to b$. For example the term $f(f(f(a)))$ is labeled as $f^\star(f^\star(f(a^\star)))$. We obtain a μTRS by forbidding rewriting inside the argument of the symbol $f^\star$; since $a^\star$ is a constant, there is nothing to be forbidden. Then for correctly labeled terms, rewriting inside redex positions is disallowed, corresponding to the outermost rewriting strategy. In order to rewrite labeled terms we have to label the rules of the TRS. By labeling R_0 with the algebra $\mathcal{A}_0$ we obtain the μTRS $\overline{R}_0$:

$$a^\star \to f(a^\star) \qquad f^\star(f(x)) \to b \qquad f^\star(f^\star(x)) \to b \qquad (\overline{R}_0)$$

which has two instances of the second rule, one for each possible value of x.

Now, despite the fact that the original TRS is outermost terminating, the transformed μTRS $\overline{R}_0$ admits an infinite rewrite sequence:

$$a^\star \to f(a^\star) \to f(f(a^\star)) \to f(f(f(a^\star))) \to \dots \tag{1}$$

The reason is that the term $f(f(a^\star))$ is not correctly labeled, as the root symbol f should have been marked. In [12] this problem is avoided by allowing labeling only with 'models'. Roughly, an algebra is a model if left- and right-hand sides of all rewrite rules have equal interpretations. However, this requirement is too strict for the purpose of marking redexes, because contraction of a redex at a position p may create a redex above p in the term tree, as witnessed by (1). In fact, for R_0 there exists *no* model which is able to distinguish between redex and non-redex positions. The rewrite step $f(a) \to f(f(a))$ creates a redex at the top. The term $f(a)$ is not a redex, and therefore its root symbol f should not

be marked. On the other hand $f(f(a))$ is a redex and so the outermost f has to be marked. The change of the labeling of a context (here $f(\Box)$) implies that the interpretation of its arguments a and $f(a)$ cannot be the same. Therefore we cannot require the rule $a \to f(a)$ to preserve the interpretation.

For this reason, we generalize the concept of model, and relax the requirement $[\ell] = [r]$ to $\exists n.\ [C[\ell]] = [C[r]]$ for all contexts C of depth n. Thus rules are allowed to change the interpretation as long as the effect is limited to contexts of a bounded depth. Algebras satisfying this weaker requirement, which we call 'C-models', are strong enough to recognize redex positions. In particular, the algebra $\mathcal{A}_0$ given above is a C-model for the system R_0. As demonstrated by the rewrite sequence (1) in the μTRS $\overline{R}_0$, for C-models it is no longer sufficient to simply label the rules: an application of the rule $a^\star \to f(a^\star)$ in the term $f(a^\star)$ creates the invalid labeled term $f(f(a^\star))$. In order to preserve correct labeling, we sometimes need to extend the rewrite rules by putting contexts around their left- and right-hand sides. Using the C-model $\mathcal{A}_0$, our algorithm transforms R_0 into the μTRS $\triangle^\pi_{\mathcal{A}_0}(R_0)$, which truthfully simulates outermost rewriting of R_0:

$$\begin{array}{cc} f(a^\star) \to f^\star(f(a^\star)) & \mathsf{top}(f^\star(f(x))) \to \mathsf{top}(b) \\ \mathsf{top}(a^\star) \to \mathsf{top}(f(a^\star)) & \mathsf{top}(f^\star(f^\star(x))) \to \mathsf{top}(b) \end{array} \qquad (\triangle^\pi_{\mathcal{A}_0}(R_0))$$

We explain this magical transformation. The rule $f(a^\star) \to f^\star(f(a^\star))$ is obtained from prepending the context $f(\Box)$ to $a \to f(a)$. This enables correct updating of the labeling of the context during rewriting. Because we still have to allow rewrite steps $a \to f(a)$ of the original TRS at the top of a term, we extend the signature with a unary function symbol top which represents the top of a term. Thus when prepending contexts we include $\mathsf{top}(\Box)$, giving rise to the rule $\mathsf{top}(a^\star) \to \mathsf{top}(f(a^\star))$. The necessity of the symbol top becomes especially apparent when considering the rule $f(f(x)) \to b$. Here prepending the context $f(\Box)$ is not even an option since $f(f(f(x))) \to f(b)$ is not an outermost rewrite step; this rule can only be applied at the top of a term. Hence we get the two rules displayed on the right, one for each possible interpretation of the variable x.

Semantic labeling has the nice property that it does not complicate termination proofs. Although semantic labeling increases the search space, termination proofs for the unlabeled system carry over to the labeled one. That is, whenever R' is a labeling of a TRS R and $\mathcal{A} = \langle A, [\cdot], \succ, \sqsupseteq \rangle$ is a monotone Σ-algebra [4] which proves termination of R, then extending $[\cdot]$ to the labeled signature Σ' by $[f^\lambda] = [f]$ for every $f \in \Sigma$ and label λ, yields a monotone Σ'-algebra which proves termination of R'. Consequently our transformation, based on a variant of semantic labeling, also does not complicate termination proofs. On the contrary, the labeled systems often allow for simpler proofs arising from the enriched signature which provides more freedom for the choice of interpretations, see [12].

Semantic labeling possibly creates extra copies of rules, and our extension might even create more copies: one for each context that has to be prepended. Despite of this fact, the implementation of our transformation in the termination prover Jambox performs efficiently on the set of examples from the Termination Problem Database (TPDB [10]).

	score	average time
Jambox	72 (93.5%)	4.1s
TrafO	46 (59.7%)	8.1s
AProVE	27 (35.0%)	10.8s

Fig. 1. Results of proving outermost termination in the competition of 2008

Jambox was best in proving termination in the category of outermost rewriting of the termination competition of 2008 [10], see Figure 1. With an average time of 4.1 seconds per termination proof, Jambox was also faster than the other participants, providing empirical evidence for the efficiency of our transformation. Not listed in Figure 1 is TTT2, which did not prove outermost termination, but performed best in disproving outermost termination.

The secret behind the efficiency of Jambox is threefold: First, we construct and minimize the algebras employed for marking redex positions, see Sections 4 and 5. Secondly, we try two labeling strategies: minimal and maximal. Minimal labeling is very efficient and contributes to 75% of the success of Jambox. In order to have a complete transformation we also employ maximal labeling. Both labelings are described in Section 6. Thirdly, we combine labeling and context extension into what we call 'dynamic labeling', where contexts are prepended depending on the interpretation of the variables. This is formalized in Section 3. All these optimizations minimize the number of rules and their size in the transformed μTRS, which is important to keep a manageable search space.

Related work. Cariboo [5] deals with outermost termination using a stand-alone approach based on induction. The very idea for a transformational approach to outermost termination comes from [8]. There the signature is enriched with unary symbols top, up, and down and the TRS is extended with 'anti-matching' rules such that $\mathsf{down}(t)$ is a redex if and only if t is not a redex with respect to the original TRS. The idea is that the symbol down is moved down in the term tree as long as no redex is encountered. Once a redex is encountered, a rewrite step is performed, and the symbol down is replaced by up, which then moves upwards again to the top of the term, marked by top. This transformation is implemented in TrafO. Based on a similarly elegant idea, Thiemann [11] defines a complete transformation from outermost to innermost rewriting, which is implemented in AProVE. For traversal to the redex positions, rules of the form $\mathsf{down}(\mathsf{isRedex}(f(\ldots))) \to f(\ldots, \mathsf{down}(\mathsf{isRedex}(\ldots)), \ldots)$ are used. In order to simulate outermost rewriting and to prevent from moving inside redexes, rules $\mathsf{isRedex}(\ell) \to \mathsf{up}(r)$ are added for every rule $\ell \to r$ of the original TRS. Then, by the innermost rewriting strategy, the latter rules have priority over the traversal rules, whenever an original redex is encountered. The simplicity of both approaches is attractive, but the yo-yoing effect in the resulting TRSs makes that the original outermost rewrite steps are 'hidden' among a vast amount of auxiliary steps. This increases derivational complexity, and makes it hard for automated termination provers to find proofs for the transformed systems.

2 Preliminaries

For a general introduction to term rewriting and to context-sensitive rewriting, the reader is referred to [9] and [6], respectively. Here we repeat some of the main definitions, for the sake of completeness, and to fix notations.

A *signature* Σ is a non-empty set of symbols each having a fixed *arity*, given by a mapping $\sharp : \Sigma \to \mathbb{N}$. Given Σ and a set $\mathcal{X}$ of variables, the *set* $Ter(\Sigma, \mathcal{X})$ *of terms over* Σ is the smallest set satisfying: $\mathcal{X} \subseteq Ter(\Sigma, \mathcal{X})$, and $f(t_1, \ldots, t_n) \in Ter(\Sigma, \mathcal{X})$ if $f \in \Sigma$ of arity n and $t_i \in Ter(\Sigma, \mathcal{X})$ for all $1 \leq i \leq n$. We use $x, y, z, \ldots$ to range over variables, and write $Var(t)$ for the set of variables occurring in a term t. Usually we leave $\mathcal{X}$ implicit and write $Ter(\Sigma)$ for the set of terms over Σ and a fixed, countably infinite set of variables $\mathcal{X}$. The set of positions $\mathcal{P}os(t) \subseteq \mathbb{N}^*$ of a term $t \in Ter(\Sigma)$ is defined as follows: $\mathcal{P}os(x) = \{\epsilon\}$ for variables $x \in \mathcal{X}$ and $\mathcal{P}os(f(t_1, \ldots, t_n)) = \{\epsilon\} \cup \{ip \mid 1 \leq i \leq \sharp f,\ p \in \mathcal{P}os(t_i)\}$. We write $t(p)$ to denote the root symbol of $t|_p$, the subterm of t rooted at p, and we write $root(t)$ for the root symbol of t, that is $root(t) = t(\epsilon)$.

A *substitution* σ is a map $\sigma : \mathcal{X} \to Ter(\Sigma, \mathcal{X})$ from variables to terms. For terms $t \in Ter(\Sigma, \mathcal{X})$ and substitutions σ, $t\sigma$ is inductively defined by $x\sigma = \sigma(x)$ if $x \in \mathcal{X}$, and $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$ otherwise. Let $\Box$ be a fresh symbol, i.e. $\Box \notin \Sigma \cup \mathcal{X}$. A *context* C is a term from $Ter(\Sigma, \mathcal{X} \cup \{\Box\})$ containing precisely one occurrence of $\Box$. By $C[s]$ we denote the term $C\sigma$ where $\sigma(\Box) = s$ and $\sigma(x) = x$ for all $x \in \mathcal{X}$. The *depth* of a context C is defined as the length $|p|$ of the position p at which $\Box$ resides, that is, the position p such that $C(p) = \Box$.

A *term rewriting system (TRS)* over Σ is a set R of pairs $\langle \ell, r\rangle \in Ter(\Sigma, \mathcal{X})^2$, called *rewrite rules* and written as $\ell \to r$, for which the *left-hand side* ℓ is not a variable ($\ell \notin \mathcal{X}$) and all variables in the *right-hand side* r occur in ℓ: $Var(r) \subseteq Var(\ell)$. For a TRS R we define $\to_R$, the *rewrite relation* induced by R as follows. For terms $s, t \in Ter(\Sigma, \mathcal{X})$ we write $s \to_R t$, or just $s \to t$ if R is clear from the context, if there exists a rule $\ell \to r \in R$, a substitution σ and a context $C \in Ter(\Sigma, \mathcal{X} \cup \{\Box\})$ such that $s = C[\ell\sigma]$ and $t = C[r\sigma]$; we sometimes write $s \to_{R,p} r$ to explicitly indicate the rewrite position p, i.e. when $C(p) = \Box$. Then s *outermost rewrites to* t at a position $p \in \mathcal{P}os(s)$, denoted by $s \xrightarrow[R,p]{\text{out}} t$, if $s \to_{R,p} t$ and for all positions p' that are a proper prefix of p: $s|_{p'}$ is not a redex.

A mapping $\mu : \Sigma \to \mathbf{2}^{\mathbb{N}}$ is called a *replacement map (for* Σ*)* if for all $f \in \Sigma$ we have $\mu(f) \subseteq \{1, \ldots, \sharp f\}$. A *context-sensitive term rewriting system (*μ*TRS)* is a pair $\langle R, \mu\rangle$ consisting of a TRS R and a replacement map μ. The set of μ*-replacing positions* $\mathcal{P}os^\mu(t)$ of a term $t \in Ter(\Sigma, \mathcal{X})$ is defined by $\mathcal{P}os^\mu(x) = \{\epsilon\}$ for $x \in \mathcal{X}$ and $\mathcal{P}os^\mu(f(t_1, \ldots, t_n)) = \{\epsilon\} \cup \{ip \mid i \in \mu(f),\ p \in \mathcal{P}os^\mu(t_i)\}$. In context-sensitive term rewriting only redexes at μ-replacing positions are contracted: s *μ-rewrites to* t, denoted $s \to_{R,\mu} t$, whenever $s \to_{R,p} t$ with $p \in \mathcal{P}os^\mu(s)$.

A *Σ-algebra* $\langle A, [\cdot]\rangle$ consists of a non-empty set A and for each n-ary $f \in \Sigma$ a function $[f] : A^n \to A$, called the *interpretation of* f. Given an *assignment* $\alpha : \mathcal{X} \to A$, the interpretation of terms $t \in Ter(\Sigma)$ is defined by: $[x, \alpha] = \alpha(x)$ and $[f(t_1, \ldots, t_n), \alpha] = [f]([t_1, \alpha], \ldots, [t_n, \alpha])$. For substitutions $\sigma : \mathcal{X} \to Ter(\Sigma, \mathcal{X})$, we write $[\sigma, \alpha]$ for the function $\lambda x.[\sigma(x), \alpha]$. For ground terms $t \in Ter(\Sigma, \varnothing)$ and substitutions $\sigma : \mathcal{X} \to Ter(\Sigma, \varnothing)$ we write $[t]$ and $[\sigma]$ for short, respectively.

3 Transformation by Dynamic Labeling

In outermost rewriting the only redexes which are allowed to be rewritten are those which are not nested within any other redex occurrence. We represent this strategy by context-sensitive rewriting by using semantic labeling: we mark the symbols which are the root of a redex in order to disallow rewriting within that redex. We first recall the definition of semantic labeling and models from [12], and then generalize these to fit our purpose.

Definition 3.1 ([12]). Let Σ be a signature. A *semantic labeling* $\langle A, [\cdot], \pi\rangle$ consists of a Σ-algebra $\langle A, [\cdot]\rangle$ and a family $\pi = \{\pi_f\}_{f\in\Sigma}$ of functions $\pi_f : A^{\sharp f} \to \Lambda_f$ where, for each $f \in \Sigma$, Λ_f is a finite and non-empty set of labels. For a term $t \in \mathit{Ter}(\Sigma)$ and $\alpha : \mathit{Var}(t) \to A$, an interpretation of its variables, we define the *labeling* $\mathit{lab}(t, \alpha)$ *of* t *with respect to* α inductively as follows:

$$\mathit{lab}(x, \alpha) = x\,,$$
$$\mathit{lab}(f(t_1, \ldots, t_n), \alpha) = f^{\pi_f([t_1,\alpha],\ldots,[t_n,\alpha])}(\mathit{lab}(t_1, \alpha), \ldots, \mathit{lab}(t_n, \alpha))\,.$$

For ground terms $t \in \mathit{Ter}(\Sigma, \varnothing)$ we just write $\mathit{lab}(t)$. Let R be a TRS over Σ. The *semantic labeling of* R is the TRS $\mathit{lab}(R)$ over the labeled signature $\mathit{lab}(\Sigma) = \{f^\lambda \mid f \in \Sigma,\ \lambda \in \Lambda_f\}$, defined by:

$$\mathit{lab}(R) = \{\mathit{lab}(\ell, \alpha) \to \mathit{lab}(r, \alpha) \mid \ell \to r \in R\,,\ \alpha : \mathit{Var}(\ell) \to A\}\,.$$

Term labeling satisfies the following useful property; see [12] for a proof.

Lemma 3.2 ([12]). *Let* $\langle A, [\cdot]\rangle$ *be a* Σ*-algebra,* $\alpha : \mathcal{X} \to A$*,* $\sigma : \mathcal{X} \to \mathit{Ter}(\Sigma)$*, and* $\overline{\sigma}(x) = \mathit{lab}(\sigma(x), \alpha)$*. Then* $\mathit{lab}(t\sigma, \alpha) = \mathit{lab}(t, [\sigma, \alpha])\overline{\sigma}$*, for all* $t \in \mathit{Ter}(\Sigma)$*.*

The Σ-algebra of a semantic labeling has to satisfy certain constraints in order to obtain that a TRS is terminating if and only if its labeled version is. In [12] the algebra has to be a 'model': A Σ-algebra $\langle A, [\cdot]\rangle$ is called a *model* of a TRS R if for all rules $\ell \to r \in R$ and assignments of variables $\alpha : \mathit{Var}(\ell) \to A$ we have that $[\ell, \alpha] = [r, \alpha]$. In the introduction we explained why this notion of model is too restrictive for our purpose. In order to be able to distinguish between redex and non-redex positions we introduce C-models, a generalization of models.

Definition 3.3. A *C-model* for a TRS R is a Σ-algebra $\langle A, [\cdot]\rangle$ such that for every rule $\ell \to r \in R$ there exists $n \in \mathbb{N}$ such that for every context C of depth n and assignment $\alpha : \mathcal{X} \to A$ we have $[C[\ell], \alpha] = [C[r], \alpha]$. When $n \in \mathbb{N}$ is minimal for a rule $\ell \to r$ with respect to this property, we call n the *C-depth for* $\ell \to r$.

As this definition suggests, it is possible to use a C-model $\mathcal{A}$ to transform a TRS R by prepending contexts to its rules in such a way that $\mathcal{A}$ becomes a model for the transformed system $\tilde{R}$, and then perform semantic labeling. But then, from the labeled version $\mathit{lab}(\tilde{R})$, every rule that contains a marked (redex) symbol within the context that was prepended to the rule in the construction of $\tilde{R}$ has to be removed again, as it would enable a rewrite step which is not outermost.

We choose for a different approach which we call 'dynamic labeling'. We stepwise extend rules by contexts, only when needed and dependent on the variable

interpretation used for the semantic labeling. For different interpretations of the variables usually different context depths are necessary for achieving equal interpretations of left- and right-hand side. In each extension step we check whether a candidate symbol is a redex symbol, and, if it is, this symbol is excluded from prepending. Here, by a redex symbol we mean a labeled symbol which indicates the presence of a redex in the original system (at the same position). Dynamic labeling is more efficient in that both the number and the size of the rules of the resulting μTRS are smaller than in the 'static' version. We explain dynamic labeling by means of the TRS R_1 consisting of the rules:

$$f(g(x)) \to f(f(g(x))) \qquad f(f(f(x))) \to x \qquad (R_1)$$

and the algebra $\mathcal{A}_1 = \langle A_1, [\cdot] \rangle$ where $A_1 = \{g, f_0, f_1, f_2\}$ and where the interpretation of the symbols is defined by $[g](x) = g$ for all $x \in A_1$, $[f](g) = f_1$ and $[f](f_i) = f_{\min(i+1,2)}$ for $i = 0, 1, 2$. Let further $\langle \mathcal{A}_1, \pi \rangle$ be the semantic labeling where π labels the symbols with the interpretations of their arguments. Then the symbols f^g and f^{f_2} are redex symbols, corresponding to redex positions with respect to the first and the second rule of R_1, respectively. The algebra $\mathcal{A}_1$ is a C-model where for the first rule the C-depth is 1, and for the second rule it is 2.

We iteratively construct sets $P_0, P_1, \ldots$, until $P_{i+1} = P_i$ for some i. The initial set P_0 consists of pairs $\langle \ell \to r, \alpha \rangle$ for each rule $\ell \to r$, and each interpretation $\alpha : \mathit{Var}(\ell) \to A_1$ of the variables. Then, in each step, P_{i+1} is obtained from P_i by replacing every pair $\langle \ell \to r, \alpha \rangle$ of P_i for which the interpretation of the left-hand side differs from the right-hand side ($[\ell, \alpha] \neq [r, \alpha]$), by the pairs $\langle C[\ell] \to C[r], \alpha' \rangle$ for every flat context C (see (2) on the next page) and every extension $\alpha' : \mathit{Var}(C[\ell]) \to A_1$ of α, such that the root of the labeled, extended left-hand side $\mathit{lab}(C[\ell], \alpha')$ is not a redex symbol. Among the flat contexts to prepend we include $\mathsf{top}(\Box)$ to cater for the case that the rule is applied at the top of the term. For R_1 the initial set P_0 is:

$$P_0 = \left\{ \langle f(g(x)) \to f(f(g(x))), \lambda x.a \rangle, \langle f(f(f(x))) \to x, \lambda x.a \rangle \mid a \in A_1 \right\}.$$

The only element $\langle \ell \to r, \alpha \rangle$ of P_0 such that $[\ell, \alpha] = [r, \alpha]$ is $\langle f(f(f(x))) \to x, \lambda x.f_2 \rangle$. For this pair no context needs to be prepended. The other pairs have to be replaced by their context extensions, and thus P_1 consists of:

$$\begin{aligned}
&\langle C[f(g(x))] \to C[f(f(g(x)))], \lambda x.a \rangle, \quad \text{for all } a \in A_1,\ C \in \{\mathsf{top}(\Box), f(\Box), g(\Box)\},\\
&\qquad \langle f(f(f(x))) \to x, \lambda x.f_2 \rangle,\\
&\langle C[f(f(f(x)))] \to C[x], \lambda x.a \rangle, \quad \text{for all } a \in A_1 \setminus \{f_2\},\ C \in \{(\mathsf{top})\Box, g(\Box)\}.
\end{aligned}$$

In the last line the context $f(\Box)$ is excluded, because the labeled left-hand side of the rule would contain the redex symbol f^{f_2} within the prepended context, and thus the step would not be outermost. Due to the outermost strategy, the original rule is only applicable under a context $C[g(\Box)]$ (where C does not contain any redexes) or at the top of a term. Now for all $(4 \cdot 3 + 1 + 3 \cdot 2 = 19)$ pairs of P_1 left- and right-hand side have equal interpretations, and the iterative construction is ended. We define $\triangle_{\mathcal{A}_1}(R_1) = P_1$, and call this set the 'dynamic context extension' of R_1 with respect to $\mathcal{A}_1$.

Secondly, the dynamic extension $\triangle_{\mathcal{A}_1}(R_1)$ is labeled using the family π which returns for each symbol f a label consisting of the interpretation of its arguments, i.e., $\pi_f(a_1, \ldots, a_{\sharp f}) = \langle a_1, \ldots, a_{\sharp f} \rangle$. Then the desired μTRS $\triangle^{\pi}_{\mathcal{A}_1}(R_1)$, which we call the *dynamic labeling of* R_1, consists of the rules $lab(\ell, \alpha) \to lab(r, \alpha)$ for every $\langle \ell \to r, \alpha \rangle \in \triangle_{\mathcal{A}_1}(R_1)$, with the replacement map μ defined by $\mu(f) = \varnothing$ if $f \in \{f^g, f^{f_2}\}$, and $\mu(f) = \{1, \ldots, \sharp f\}$ otherwise, for all $f \in lab(\Sigma)$.

We now formalize dynamic labeling. For the remainder of this section we fix an arbitrary TRS R over Σ, and let $\mathcal{A} = \langle A, [\cdot] \rangle$ be a C-model for R. We assume top to be a fresh symbol ($\mathsf{top} \notin \Sigma$) representing the top of a term, and abbreviate $\Sigma_{\mathsf{top}} = \Sigma \cup \{\mathsf{top}\}$. We extend $\mathcal{A}$ to a Σ_{top}-algebra by choosing an arbitrary but fixed element $a \in A$ and defining the interpretation of top as the constant function $[\mathsf{top}] = \lambda x.a$. Furthermore, we let $\langle \mathcal{A}, \pi \rangle$ be a semantic labeling, and $\Sigma^{red} \subseteq lab(\Sigma)$ a subset of the labeled signature, called the set of *redex symbols*. The definition makes use of *flat contexts fresh for* $t \in Ter(\Sigma_{\mathsf{top}})$:

$$\mathcal{C}^{\flat}_t = \left\{ f(x_1, \ldots, x_{j-1}, \Box, x_{j+1}, \ldots, x_{\sharp f}) \mid f \in \Sigma_{\mathsf{top}},\ j \in \{1, \ldots, \sharp f\} \right\} \quad (2)$$

where $x_1, x_2, \ldots \in \mathcal{X}$ such that $x_i \notin Var(t)$. Furthermore, for partial functions $f, g : S \rightharpoonup T$ with disjoint domains, we write $f+g$ for the *union of* f *and* g, defined by $(f + g)(x) = f(x)$ if $x \in dom(f)$, and $(f + g)(x) = g(x)$ if $x \in dom(g)$.

Definition 3.4. The *dynamic context extension of* R, denoted by $\triangle_{\mathcal{A}}(R)$, is defined as the fixed point of the following construction of sets $P_0, P_1, \ldots$, that is, $\triangle_{\mathcal{A}}(R) = P_i$ as soon as $P_{i+1} = P_i$ for some i. The initial set is defined by:

$$P_0 = \left\{ \langle \ell \to r, \alpha \rangle \mid \ell \to r \in R,\ \alpha : Var(\ell) \to A \right\},$$

and for $i = 0, 1, \ldots$ the set P_{i+1} is obtained from P_i by replacing every pair $\langle \ell \to r, \alpha \rangle$ such that $[\ell, \alpha] \neq [r, \alpha]$, or $r \in \mathcal{X}$,[1] by all pairs in $\triangle(\ell \to r, \alpha)$ where:

$$\triangle(\ell \to r, \alpha) = \left\{ \langle C[\ell] \to C[r], \alpha + \beta \rangle \;\middle|\; C \in \mathcal{C}^{\flat}_{\ell},\ \beta : Var(C) \to A,\ root(lab(C[\ell], \alpha + \beta)) \notin \Sigma^{red} \right\}.$$

Then, the *dynamic labeling of* R is the μTRS $\langle \triangle^{\pi}_{\mathcal{A}}(R), \mu \rangle$ consisting of:

$$\triangle^{\pi}_{\mathcal{A}}(R) = \left\{ lab(\ell, \alpha) \to lab(r, \alpha) \mid \langle \ell \to r, \alpha \rangle \in \triangle_{\mathcal{A}}(R) \right\},$$

and the replacement map μ, defined by $\mu(f) = \varnothing$ if $f \in \Sigma^{red}$, and $\mu(f) = \{1, \ldots, \sharp f\}$ otherwise, for all $f \in lab(\Sigma_{\mathsf{top}})$. Whenever the set Σ^{red}, which determines the replacement map, is clear from the context, we write $\triangle^{\pi}_{\mathcal{A}}(R)$ as a shorthand for $\langle \triangle^{\pi}_{\mathcal{A}}(R), \mu \rangle$.

Notice that the construction of $\triangle_{\mathcal{A}}(R)$ is guaranteed to terminate because of the assumption that $\mathcal{A}$ is a C-model.

We come to the first main theorem, stating that outermost ground termination of R is implied by termination of the transformed system $\triangle^{\pi}_{\mathcal{A}}(R)$.

[1] The condition $r \notin \mathcal{X}$ eliminates collapsing rules. This is used in the proof of Thm. 6.5, which states completeness. Without this condition, the transformation is still sound (Thm. 3.7). Nota bene: in the TRS R_1 worked out before, $r \notin \mathcal{X}$ is not used.

For the remainder of this section, we assume that the set $\Sigma^{red} \subseteq lab(\Sigma)$ contains redex symbols only, that is, for all ground terms t and $p \in \mathcal{P}os(t)$:

$$lab(t)(p) \in \Sigma^{red} \text{ implies that } t|_p \text{ is a redex with respect to } R \qquad (\ddagger)$$

Lemma 3.5. *Let $s, t \in Ter(\Sigma, \varnothing)$ be ground terms and $p \in \mathcal{P}os(s)$ such that $s \xrightarrow{out}_{R,p} t$. Then for all proper prefixes q of $1p$ we have $lab(\mathsf{top}(s))(q) \notin \Sigma^{red}$.*

Proof. If $q = \epsilon$, this follows from $\mathsf{top}^{\lambda} \notin \Sigma^{red}$ for any label λ. If $q \neq \epsilon$, then $lab(\mathsf{top}(s))(q) = lab(s)(q')$ with q' a proper prefix of p, and if $lab(s)(q') \in \Sigma^{red}$, then by assumption $(\ddagger)$ the term s contains a redex at position q', quod non. □

The following lemma states that any outermost ground rewrite step in R can be transformed into a rewrite step in $\triangle^{\pi}_{\mathcal{A}}(R)$. For ground substitutions $\sigma : \mathcal{X} \to Ter(\Sigma, \varnothing)$ define $\overline{\sigma}(x) = lab(x\sigma)$.

Lemma 3.6. *Let $s, t \in Ter(\Sigma, \varnothing)$ be ground terms such that $s \xrightarrow{out}_R t$. Then:*

$$lab(\mathsf{top}(s)) \to_{\triangle^{\pi}_{\mathcal{A}}(R)} lab(\mathsf{top}(t)) \,.$$

Proof. Assume $s \xrightarrow{\text{out}}_{R,p} t$ for some position $p \in \mathcal{P}os(s)$. Then there exists a rule $\ell \to r \in R$, a context C with $C(p) = \Box$ and a ground substitution σ such that $s = C[\ell\sigma]$ and $t = C[r\sigma]$. We consider the construction of the dynamic context extension from Definition 3.4, and prove by induction that for all $i = 0, 1, \ldots$ there exists a context C_i which is a prefix of $\mathsf{top}(C)$, a ground substitution σ_i, and terms ℓ_i, r_i such that $\mathsf{top}(s) = C_i[\ell_i\sigma_i]$, $\mathsf{top}(t) = C_i[r_i\sigma_i]$ and $\langle \ell_i \to r_i, [\sigma_i] \rangle \in P_i$. For the base case we have $\langle \ell_0 \to r_0, [\sigma_0] \rangle \in P_0$ with $\ell_0 = \ell$, $r_0 = r$, $\sigma_0 = \sigma$, and $C_0 = \mathsf{top}(C)$. For the induction step we assume the existence of C_i, σ_i, and $\langle \ell_i \to r_i, [\sigma_i] \rangle \in P_i$ with the above properties. If $[\ell_i, [\sigma_i]] = [r_i, [\sigma_i]]$ and $r_i \notin \mathcal{X}$ then by definition $\langle \ell_i \to r_i, [\sigma_i] \rangle \in P_{i+1}$, and so we are done. For the remaining cases $[\ell_i, [\sigma_i]] \neq [r_i, [\sigma_i]]$ and $r_i \in \mathcal{X}$, we first show that $C_i \neq \Box$. If $[\ell_i, [\sigma_i]] \neq [r_i, [\sigma_i]]$ and $C_i = \Box$, then $\ell_i\sigma_i = \mathsf{top}(s)$ and $r_i\sigma_i = \mathsf{top}(t)$, and hence $root(\ell_i) = root(r_i) = \mathsf{top}$, contradicting $[\ell_i, [\sigma_i]] \neq [r_i, [\sigma_i]]$ (recall that the interpretation of top is constant). Furthermore, we have $r_i \in \mathcal{X}$ only if $i = 0$, and then $C_i = \mathsf{top}(C) \neq \Box$. Thus we have $C_i = D[D'\sigma']$ for some context D, flat context $D' \in \mathcal{C}^{\flat}_{\ell_i}$ and substitution σ'. We choose $C_{i+1} = D$, $\ell_{i+1} = D'[\ell_i]$, $r_{i+1} = D'[r_i]$, and $\sigma_{i+1} = \sigma_i + \sigma'$. It remains to be shown that $\langle \ell_{i+1} \to r_{i+1}, [\sigma_{i+1}] \rangle \in P_{i+1}$. For this it suffices to prove that $root(lab(\ell_{i+1}, [\sigma_{i+1}])) \notin \Sigma^{red}$. We have $C_{i+1}[\ell_{i+1}\sigma_{i+1}] = \mathsf{top}(s)$. Let q be the position such that $C_{i+1}(q) = \Box$. Then, by Lemma 3.2 we obtain $root(lab(\ell_{i+1}, [\sigma_{i+1}])) = root(lab(\ell_{i+1}\sigma_{i+1})) = \mathsf{top}(lab(s))(q)$. Note that q is a proper prefix of $1p$, and so $lab(s)(q') \notin \Sigma^{red}$ by Lemma 3.5.

Let $i \in \mathbb{N}$ be such that $P_{i+1} = P_i$. By the result above we have $\langle \ell_i \to r_i, [\sigma_i] \rangle \in \triangle_{\mathcal{A}}(R)$, with $[\ell_i\sigma_i] = [r_i\sigma_i]$, and then $lab(\ell_i, [\sigma_i]) \to lab(r_i, [\sigma_i]) \in \triangle^{\pi}_{\mathcal{A}}(R)$ by definition. Let τ and υ be defined by $\tau(\Box) = \ell_i\sigma_i$, $\upsilon(\Box) = r_i\sigma_i$, and $\tau(x) = \upsilon(x) = x$ for $x \in \mathcal{X}$. Then we have that $lab(C_i, [\tau]) = lab(C_i, [\upsilon])$ since $[\tau] = [\upsilon]$. Let $E = lab(C_i, [\tau])$. We get $lab(\mathsf{top}(s)) = lab(C_i[\ell_i\sigma_i]) = lab(C_i\tau) = E\overline{\tau} = E[lab(\ell_i\sigma_i)] = E[lab(\ell_i, [\sigma_i])\overline{\sigma_i}]$ and $lab(\mathsf{top}(t)) = \ldots = E[lab(r_i, [\sigma_i])\overline{\sigma_i}]$, by Lemma 3.2. Then, by Lemma 3.5 we have $lab(\mathsf{top}(s)) \to_{\triangle^{\pi}_{\mathcal{A}}(R)} lab(\mathsf{top}(t))$. □

Theorem 3.7. *R is outermost ground terminating if $\triangle^{\pi}_{\mathcal{A}}(R)$ is terminating.*

Proof. Assume R admits an infinite outermost rewrite sequence $t_1 \overset{\text{out}}{\to}_R t_2 \overset{\text{out}}{\to}_R t_3 \overset{\text{out}}{\to}_R \ldots$. Then from Lemma 3.6 it follows that $\triangle^{\pi}_{\mathcal{A}}(R)$ allows an infinite rewrite sequence: $lab(\mathsf{top}(t_1)) \to_{\triangle^{\pi}_{\mathcal{A}}(R)} lab(\mathsf{top}(t_2)) \to_{\triangle^{\pi}_{\mathcal{A}}(R)} lab(\mathsf{top}(t_3)) \to_{\triangle^{\pi}_{\mathcal{A}}(R)} \ldots$. □

Theorem 3.7 is about outermost ground termination. This is not a restriction because by adding a fresh constant 0 and a fresh unary symbol s, outermost ground termination coincides with outermost termination:

Lemma 3.8. *The TRS R over the signature Σ is outermost terminating if and only if R over the signature $\Sigma \cup \{s, 0\}$ is outermost ground terminating.* □

4 Constructing Suitable Algebras

In this section we construct C-models that are able to recognize redex positions with respect to left-linear rules. The construction of C-models is similar to the construction of a deterministic tree automaton (DTA) for recognizing left-linear redexes. A DTA is a Σ-algebra $\langle A, [\cdot]\rangle$ with a distinguished set $A_F \subseteq A$ of final states. A term t is accepted by the automaton whenever $[t] \in A_F$. The difference with the construction of a DTA is that for the construction of a C-model we do not distinguish final and non-final states, but instead have a family of functions $isRedex_f : A^{\sharp f} \to Bool$ for indicating the presence of a redex.

Definition 4.1. A *redex-algebra* $\mathcal{A} = \langle A, [\cdot], isRedex\rangle$ is a Σ-algebra $\langle A, [\cdot]\rangle$ with a family $\{isRedex_f\}_{f\in\Sigma}$ of functions $isRedex_f : A^{\sharp f} \to Bool$. The *language of* $\mathcal{A}$ is the set $\mathcal{L}(\mathcal{A}) = \{f(t_1, \ldots, t_n) \in Ter(\Sigma, \varnothing) \mid isRedex_f([t_1], \ldots, [t_n]) = true\}$.

By this separation of tasks our approach allows for smaller algebras, because, intuitively, the algebra needs to 'remember' only the subterms $t_1,\ldots,t_n$ and not $f(t_1, \ldots, t_n)$ itself. To see this, consider the single-rule system:

$$f(g(x)) \to a\,.$$

A tree automaton recognizing redex positions for this TRS needs at least three states: one for indicating a redex $f(g(\ldots))$, one for $g(\ldots)$, and one garbage state. For redex-algebras two states suffice: one state for $g(\ldots)$ and one for garbage. Then $isRedex_f(g(\ldots)) = true$ and $false$, otherwise.

The 'core' of a redex-algebra consists of all interpretations of ground terms:

Definition 4.2. Let $\mathcal{A} = \langle A, [\cdot], isRedex\rangle$ be a redex-algebra over Σ. The *core of* $\mathcal{A}$ is the redex-algebra $\mathcal{A}_c = \langle A_c, [\cdot]_c, isRedex_c\rangle$ where A_c is the smallest set such that $[f](a_1, \ldots, a_n) \in A_c$ whenever $f \in \Sigma$ and $a_1, \ldots, a_n \in A_c$, and where $[\cdot]_c$ and $isRedex_c$ are the restrictions of $[\cdot]$ and $isRedex$ to A_c, respectively. Furthermore we say that $\mathcal{A}$ *is core* whenever $\mathcal{A} = \mathcal{A}_c$.

Lemma 4.3. *Let $\mathcal{A} = \langle A, [\cdot], isRedex\rangle$ be a redex-algebra over Σ. Then for every $a \in \mathcal{A}_c$ there exists a ground term $t \in Ter(\Sigma, \varnothing)$ with $[t] = a$.* □

We now describe a syntactical construction of redex-algebras. The idea is to build Σ-algebras that 'remember' proper subterms of left-hand sides. Given this interpretation, the *isRedex* functions decide whether a redex is present. We first define some auxiliary functions.

Let $\bot$ be a fresh symbol, $\bot \notin \Sigma$, and define $\mathcal{T} = \mathit{Ter}(\Sigma \cup \{\bot\}, \varnothing)$. The function $\mathit{cut} : \mathit{Ter}(\Sigma, \mathcal{X}) \to \mathcal{T}$ is defined such that $\mathit{cut}(t)$ is the result of replacing all variables in t by $\bot$. We define $\mathit{match} : \mathcal{T} \times \mathcal{T} \to \mathit{Bool}$ such that $\mathit{match}(s,t) = \mathit{true}$ if s can be obtained from t by replacing subterms of t by $\bot$, and $\mathit{match}(s,t) = \mathit{false}$, otherwise. Let further $\mathit{merge}(s,t)$ be the 'most general common instance' of s and t, that is, $\mathit{merge} : \mathcal{T} \times \mathcal{T} \rightharpoonup \mathcal{T}$ is defined by: $\mathit{merge}(\bot, t) = t$, $\mathit{merge}(t, \bot) = t$, and $\mathit{merge}(f(s_1, \ldots, s_n), f(t_1, \ldots, t_n)) = f(\mathit{merge}(s_1, t_1), \ldots, \mathit{merge}(s_n, t_n))$, and undefined whenever there exists a position $p \in \mathcal{P}os(s)$ such that $s(p) \in \Sigma$, $t(p) \in \Sigma$, and $s(p) \neq t(p)$. Finally we define $\mathit{shrink} : \mathcal{T} \times \mathbf{2}^{\mathcal{T}} \to \mathcal{T}$ such that $\mathit{shrink}(s, T)$ is the largest $t \in T$ (with respect to the number of symbols) such that $\mathit{match}(t, s)$ is *true*. Note that $\mathit{shrink}(s, T)$ is well-defined whenever T is closed under *merge* and $\bot \in T$: whenever two terms $t_1 \neq t_2$ of equal size match s then $\mathit{merge}(t_1, t_2)$ is larger and matches s.

Definition 4.4. Let R be a TRS. The *redex-algebra for* R is the core of the redex-algebra $\langle A, [\cdot], \mathit{isRedex} \rangle$, where A is the smallest set such that $\bot \in A$ and

- $t \in A$ for every proper subterm t of $\mathit{cut}(\ell)$ with ℓ a linear left-hand side of R,
- $\mathit{merge}(s,t) \in A$ whenever $s, t \in A$ and $\mathit{merge}(s,t)$ is defined.

Then $[\cdot]$ is defined by $[f](t_1, \ldots, t_n) = \mathit{shrink}(f(t_1, \ldots, t_n), A)$. And, for every $f \in \Sigma$ we define $\mathit{isRedex}_f(t_1, \ldots, t_n) = \mathit{true}$ if $f(t_1, \ldots, t_n)$ is an instance of a linear left-hand side of R, and $\mathit{isRedex}_f(t_1, \ldots, t_n) = \mathit{false}$, otherwise.

Example 4.5. Consider the term rewriting system R consisting of the rules:

$$c(c(c(x))) \to a\,, \qquad\qquad c(c(a)) \to c(c(c(c(a))))\,.$$

By Definition 4.4 we obtain $A = \{c(c(\bot)), c(\bot), \bot, c(a), a\}$. Here $[a] = a$, $[c](a) = c(a)$, $[c](c(a)) = c(c(\bot))$ and $[c](c(c(\bot))) = c(c(\bot))$. Thus the elements $\bot$ and $c(\bot)$ are not part of the core and hence not of the redex-algebra for the TRS. Here we have $\mathit{isRedex}_c(c(a)) = \mathit{isRedex}_c(c(c(\bot))) = \mathit{true}$, and *false* otherwise.

Example 4.6. We compute the domain of the redex-algebra for the TRS:

$$\begin{aligned} f(x,y) &\to a(f(c(x), y))\,, & a(f(c(c(x)), y)) &\to e\,,\\ f(x,y) &\to b(f(x, c(y)))\,, & b(f(x, c(c(y)))) &\to e\,. \end{aligned}$$

The subterms of $\mathit{cut}(\ell)$ of linear left-hand sides ℓ are: $S = \{\, \bot,\ f(c(c(\bot)), \bot),\ f(\bot, c(c(\bot))),\ c(c(\bot)),\ c(\bot)\,\}$. The closure of S under *merge* yields the domain: $A = S \cup \{f(c(c(\bot)), c(c(\bot)))\}$.

Lemma 4.7. *Let R be a TRS over Σ and $\mathcal{A}$ the redex-algebra for R. Then for all ground terms $t \in \mathit{Ter}(\Sigma, \varnothing)$ we have $t \in \mathcal{L}(\mathcal{A})$ if and only if t is a redex with respect to a left-linear rule in R.*

The proof proceeds by induction over the term structure. The 'only if'-part is crucial for soundness of our transformation, whereas the 'if'-part is needed for completeness for left-linear TRSs.

5 Minimizing Algebras

In this section we are concerned with the minimization of redex-algebras. The algorithm is similar to the minimization of deterministic tree automata, see [3]. For the set of 291 TRSs of the outermost termination competition of 2008 [10], the redex-algebras constructed according to Definition 4.4 have an average size of 4.6 elements. After an application of the minimization algorithm described here, the average size falls to 3.4, a reduction of 27%. This reduction has a polynomial influence on the number of rules of the transformed system.

Definition 5.1. Core redex-algebras $\mathcal{A}_1$, $\mathcal{A}_2$ are *equivalent* if $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$.

For a given core redex-algebra we now construct a minimal equivalent algebra. The difference to the minimization of tree automata from [3] lies in the initial equivalence E_0. For tree automata E_0 consists of two partitions, the final and the non-final states. In our setting two states are initially equivalent if they cannot be distinguished using the *isRedex* functions. In general this can yield any number of partitions between 1 and $|A|$.

Definition 5.2. Let $\mathcal{A} = \langle A, [\cdot], \mathit{isRedex}\rangle$ be a core redex-algebra over Σ. We define equivalence relations E_i for $i \in \mathbb{N}$ on the elements of A. Initially two elements $a, b \in A$ are equivalent, $a\ E_0\ b$, if $\mathit{isRedex}_f(\boldsymbol{x}, a, \boldsymbol{y}) = \mathit{isRedex}_f(\boldsymbol{x}, b, \boldsymbol{y})$ for all symbols $f \in \Sigma$, $j \in \{1, \ldots, \sharp f\}$, $\boldsymbol{x} \in A^{j-1}$, and $\boldsymbol{y} \in A^{\sharp f - j}$. Then for $i = 0, 1, \ldots$ and $a, b \in A$ we define $a\ E_{i+1}\ b$ to be the conjunction of $a\ E_i\ b$ and $[f](\boldsymbol{x}, a, \boldsymbol{y})\ E_i\ [f](\boldsymbol{x}, b, \boldsymbol{y})$ for all $f \in \Sigma$, $j \in \{1, \ldots, \sharp f\}$, $\boldsymbol{x} \in A^{j-1}$, and $\boldsymbol{y} \in A^{\sharp f - j}$. We stop when $E_{i+1} = E_i$ for some $i \in \mathbb{N}$. Then we define $E = E_i$.

For $a \in A$ we use $\lceil a \rceil$ to denote the equivalence class of a with respect to E. The *minimized redex-algebra* is defined by $\mathcal{A}^{min} = \langle E, [\cdot]^E, \mathit{isRedex}^E\rangle$ where for every $f \in \Sigma$ we define $[f]^E : E^{\sharp f} \to E$ by $[f]^E(\lceil a_1 \rceil, \ldots, \lceil a_n \rceil) = \lceil f(a_1, \ldots, a_n) \rceil$, and $\mathit{isRedex}_f^E : E^{\sharp f} \to \mathit{Bool}$ by $\mathit{isRedex}_f^E(\lceil a_1 \rceil, \ldots, \lceil a_n \rceil) = \mathit{isRedex}_f(a_1, \ldots, a_n)$.

Lemma 5.3. *Let $\mathcal{A}$ be a core redex-algebra, then $\mathcal{A}$ is equivalent to $\mathcal{A}^{min}$.* □

Example 5.4. We consider the TRS R consisting of the following three rules:

$$f(i(a)) \to a\,, \qquad f(j(a)) \to a\,, \qquad f(a) \to a\,.$$

The redex-algebra for R is $A = \{a, i(a), j(a), \bot\}$ with the interpretation $[a] = a$, $[i](a) = i(a)$, $[j](a) = j(a)$, and the interpretation is $\bot$ in all non-listed cases; $\mathit{isRedex}_f(x) = \mathit{true}$ for all $x \neq \bot$, and *false*, otherwise.

The minimization algorithm starts with $E_0 = \{\{a, i(a), j(a)\}, \{\bot\}\}$ as initial equivalence, since $\bot$ can be distinguished from the other elements due to $[f](\bot) = \mathit{false}$. The first iteration of the algorithm yields $E_1 = \{\{a\}, \{i(a), j(a)\}, \{\bot\}\}$ as

$[i](a) = i(a)$ whereas $[i](i(a)) = [i](j(a)) = \bot$. The elements $i(a)$ and $j(a)$ are indistinguishable, and so in the second iteration we obtain $E_2 = E_1$. Thus the elements $i(a)$ and $j(a)$ are identified and we obtain an algebra that has one element less than the algebra we started with.

6 Two Versions of Dynamic Labeling

In the previous sections we have constructed and minimized redex-algebras for recognizing redex positions. For completing the transformation we still need to define how the symbols are labeled. In this section we introduce two labelings that arise naturally: minimal and maximal labeling. In minimal labeling symbols are marked with a $\star$ if they correspond to redex positions and stay unlabeled otherwise. This labeling creates a small signature and thereby results in a small number of rules of the transformed system.

In the sequel we fix R to be a TRS over Σ and $\langle \mathcal{A}, \mathit{isRedex}\rangle$ with $\mathcal{A} = \langle A, [\cdot]\rangle$ the minimized redex-algebra for R.

Definition 6.1. The *minimal labeling* for R is the semantic labeling $\langle \mathcal{A}, \pi\rangle$ defined for every $f \in \Sigma_{\mathsf{top}}$ by $\pi_f(a_1, \ldots, a_{\sharp f}) = \star$ if $\mathit{isRedex}_f(a_1, \ldots, a_{\sharp f}) = \mathit{true}$, and $\pi_f(a_1, \ldots, a_{\sharp f}) = \epsilon$, otherwise; the redex symbols are $\Sigma^{red} = \{f^\star \mid f \in \Sigma\}$.

Theorem 6.2. *Let $\langle \mathcal{A}, \pi\rangle$ and Σ^{red} be the minimal labeling for R. Then R is outermost ground terminating if $\triangle^{\pi}_{R}(\mathcal{A})$ is terminating.*

Proof. An application of Theorem 3.7 together with Lemmas 4.7 and 5.3. □

Minimal labeling is sound and efficient, but it is not complete:

Example 6.3. The following term rewriting system is outermost terminating:

$$\mathit{inf}(x) \to \mathit{cons}(x, \mathit{inf}(s(x))) \qquad \mathit{cons}(s(x), y) \to \mathit{nil} \qquad (R_2)$$

The minimized redex-algebra for R_2 is $\mathcal{A}_2 = \langle A_2, [\cdot]\rangle$ with $A_2 = \{s, \bot\}$, $[s](x) = s$, $[\mathit{inf}](x) = [\mathit{cons}](x, y) = \bot$, and $[\mathit{nil}] = \bot$. With minimal labeling we have $\pi_{\mathit{cons}}(s, x) = \star$ and $\pi_{\mathit{inf}}(x) = \star$ for all $x \in A_2$, and unmarked (ϵ) otherwise. The dynamic labeling $\triangle^{\pi}_{\mathcal{A}_2}(R_2)$ of R_2 w.r.t. $\langle \mathcal{A}_2, \pi\rangle$ consists of the following rules, the first two of which arise from the *inf*-rule, with $\alpha(x) = \bot$, and $\alpha(x) = s$ resp.:

$$\begin{aligned} \mathit{inf}^\star(x) &\to \mathit{cons}(x, \mathit{inf}^\star(s(x)))\,, \\ \mathit{inf}^\star(x) &\to \mathit{cons}^\star(x, \mathit{inf}^\star(s(x)))\,, \\ \mathit{cons}^\star(s(x), y) &\to \mathit{nil}\,. \end{aligned}$$

with $\mu(\mathit{inf}^\star) = \mu(\mathit{cons}^\star) = \varnothing$. But now $\triangle^{\pi}_{\mathcal{A}_2}(R_2)$ admits an infinite derivation:

$$\mathit{inf}^\star(x) \to \mathit{cons}(x, \mathit{inf}^\star(s(x))) \to \mathit{cons}(x, \mathit{cons}(s(x), \mathit{inf}^\star(s(s(x))))) \to \ldots\,.$$

The third term is labeled incorrectly, as the inner *cons* should be marked. The reason is that in the second step, instead of the first $\mathit{inf}^\star$-rule, the second should have been applied; however, the left-hand side $\mathit{inf}^\star(x)$ contains too little information to 'decide' what the labeling of the right-hand side should be.

This motivates the use of maximal labeling for which correct labeling is preserved under rewriting. Symbols are labeled with the interpretation of their arguments:

Definition 6.4. The *maximal labeling* for R is the semantic labeling $\langle \mathcal{A}, \pi \rangle$ defined for every $f \in \Sigma_{\mathsf{top}}$ by: $\Lambda_f = A^{\sharp f}$, $\pi_f(a_1, \ldots, a_{\sharp f}) = \langle a_1, \ldots, a_{\sharp f} \rangle$ together with the redex symbols $\Sigma^{red} = \{f^{\langle a_1,\ldots,a_{\sharp f}\rangle} \mid \mathit{isRedex}_f(a_1, \ldots, a_{\sharp f}) = \mathit{true}\}$.

Maximal labeling is sound for all TRSs, and complete for quasi-left linear TRSs. A TRS R is called *quasi left-linear* if every non-linear left-hand side of a rule in R is an instance of a linear left-hand side from R.

Theorem 6.5. *Let $\langle \mathcal{A}, \pi \rangle$ with Σ^{red} be the maximal labeling for R. Then R is outermost ground terminating if $\triangle^{\pi}_R(\mathcal{A})$ is terminating. Moreover, if R is quasi left-linear, the reverse direction holds as well.*

Proof. The first claim is a consequence of Theorem 3.7 and Lemmas 4.7 and 5.3. For the second claim, let R be quasi left-linear. Assume that R is outermost terminating but $\triangle^{\pi}_{\mathcal{A}}(R)$ is not terminating. We introduce types for $\triangle^{\pi}_{\mathcal{A}}(R)$ over the sorts $A \cup \{\mathsf{top}\}$. For every $f^{\lambda} \in \mathit{lab}(\Sigma_{\mathsf{top}})$ with $\lambda = \langle a_1, \ldots, a_{\sharp f} \rangle$ we define f^{λ} to have input sorts $\langle a_1, \ldots, a_n \rangle$ and output sort $[f](a_1, \ldots, a_n)$, except for top for which we fix output sort top. An adaptation of [7, Proposition 5.5.24] for μTRSs together with non-collapsingness of $\triangle^{\pi}_{\mathcal{A}}(R)$ yields the existence of a well-sorted infinite $\triangle^{\pi}_R(\mathcal{A})$ rewrite sequence τ. By Lemma 4.3 we have a ground term for every sort in A. Thus by applying a ground substitution to τ we get a well-sorted infinite ground term rewrite sequence τ'. Well-sortedness implies correct labeling: for every well-sorted term $t \in \mathit{Ter}(\mathit{lab}(\Sigma_{\mathsf{top}}), \varnothing)$ there exists a term $t' \in \mathit{Ter}(\Sigma_{\mathsf{top}}, \varnothing)$ such that $t = \mathit{lab}(t')$. Moreover, by well-sortedness the symbol top can only occur at the top of a term; without loss of generality we assume that every term in τ' has top as root. Hence to arrive at a contradiction it suffices to show that for all terms $s, t \in \mathit{Ter}(\Sigma, \varnothing)$ with $\mathit{lab}(\mathsf{top}(s)) \to_{\triangle^{\pi}_R(\mathcal{A})} \mathit{lab}(\mathsf{top}(t))$ we have $s \xrightarrow{\text{out}}_R t$. By construction, every rule in $\triangle^{\pi}_R(\mathcal{A})$ is a context extension plus labeling of a rule in R. Let $\rho : s \to_R t$ be the corresponding step. What remains to be shown is that ρ is an outermost step. Assume there would be a redex u above the rewrite position. Then by Lemma 4.7 we have $u \in \mathcal{L}(\mathcal{A}_R)$ where $\mathcal{A}_R$ is the redex-algebra for R (Definition 4.4). From Lemma 5.3 it follows that $u \in \mathcal{L}(\mathcal{A})$ as $\mathcal{A}$ is the minimization of $\mathcal{A}_R$. By definition of maximal labeling we get $\mathit{root}(\mathit{lab}(u)) \in \Sigma^{red}$. But then this symbol must be in $\mathit{lab}(\mathsf{top}(s))$, either above the applied $\triangle^{\pi}_R(\mathcal{A})$ rule or within the prepended context. Both cases yield a contradiction: the former since $\mu(\mathit{root}(\mathit{lab}(u))) = \varnothing$ would prohibit the μ-step, and the latter since we do not prepend symbols from Σ^{red}. □

Example 6.6. We revisit Example 6.3, but this time we use maximal labeling:

$$\mathit{inf}^{\perp}(x) \to \mathit{cons}^{\perp,\perp}(x, \mathit{inf}^{s}(s^{\perp}(x))) \,, \qquad \mathit{inf}^{s}(x) \to \mathit{cons}^{s,\perp}(x, \mathit{inf}^{s}(s^{s}(x))) \,,$$
$$\mathit{cons}^{s,\perp}(s^{\perp}(x), y) \to \mathit{nil} \,, \qquad \mathit{cons}^{s,\perp}(s^{s}(x), y) \to \mathit{nil} \,,$$
$$\mathit{cons}^{s,s}(s^{\perp}(x), y) \to \mathit{nil} \,, \qquad \mathit{cons}^{s,s}(s^{s}(x), y) \to \mathit{nil} \,,$$

with $\mu(\mathit{inf}^{\perp}) = \mu(\mathit{inf}^{s}) = \mu(\mathit{cons}^{s,\perp}) = \mu(\mathit{cons}^{s,s}) = \varnothing$. This μTRS is indeed terminating as opposed to the μTRS constructed in Example 6.3.

7 Discussion

For arbitrary TRSs our transformation (including the construction of C-models) is sound, and for quasi left-linear TRSs it is complete (see Theorem 6.5). The redex-algebra we construct recognizes redexes with respect to left-linear rules. As a consequence, in the μTRS $\triangle^{\pi}_{\mathcal{A}}(R)$ rewriting is forbidden only inside such redex positions. This corresponds to a weakening of the outermost rewriting strategy: contraction of redexes is allowed as long as they are not strictly contained in a redex occurrence with respect to a left-linear rule. We stress that our transformation with maximal labeling is complete for termination with respect to this rewrite strategy for all TRSs.

An open question is whether there are interesting labelings between minimal and maximal. In particular, are there more efficient complete labelings? Here efficiency is measured in the size of the signature and the number of rules of the transformed system. In Example 6.6 it would have been sufficient to label *cons* with the interpretation of the left argument, saving two symbols and two rules of the transformed system.

References

1. Alarcón, B., Emmes, F., Fuhs, C., Giesl, J., Gutiérrez, R., Lucas, S., Schneider-Kamp, P., Thiemann, R.: Improving Context-Sensitive Dependency Pairs. In: LPAR 2008. LNCS, vol. 5330, pp. 636–651. Springer, Heidelberg (2008)
2. Arts, T., Giesl, J.: Termination of Term Rewriting Using Dependency Pairs. Theoretical Computer Science 236, 133–178 (2000)
3. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2007), `http://www.grappa.univ-lille3.fr/tata`
4. Endrullis, J., Waldmann, J., Zantema, H.: Matrix Interpretations for Proving Termination of Term Rewriting. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 574–588. Springer, Heidelberg (2006)
5. Fissore, O., Gnaedig, I., Kirchner, H.: Cariboo, a Termination Proof Tool for Rewriting-Based Programming Languages with Strategies, Version 1.0 (2004)
6. Lucas, S.: Context-Sensitive Computations in Functional and Functional Logic Programs. Journal of Functional and Logic Programming 1998(1) (1998)
7. Ohlebusch, E.: Advanced Topics in Term Rewriting. Springer, New York (2002)
8. Raffelsieper, M., Zantema, H.: A Transformational Approach to Prove Outermost Termination Automatically. ENTCS 237, 3–21 (2009)
9. Terese: Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press, Cambridge (2003)
10. Termination Competition, `http://www.termination-portal.org/`
11. Thiemann, R.: From Outermost Termination to Innermost Termination. In: SOFSEM 2009, pp. 533–545 (2009)
12. Zantema, H.: Termination of Term Rewriting by Semantic Labelling. Fundamenta Informaticae 24, 89–105 (1995)

A Fully Abstract Semantics for Constructor Systems*

Francisco Javier López-Fraguas, Juan Rodríguez-Hortalá, and Jaime Sánchez-Hernández

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
fraguas@sip.ucm.es, jrodrigu@fdi.ucm.es, jaime@sip.ucm.es

Abstract. Constructor-based term rewriting systems are a useful subclass of TRS, in particular for programming purposes. In this kind of systems constructors determine a universe of values, which are the expected output of the computations. Then it would be natural to think of a semantics associating each expression to the set of its reachable values. Somehow surprisingly, the resulting semantics has poor properties, for it is not compositional nor fully abstract when non-confluent systems are considered. In this paper we propose a novel semantics for expressions in constructor systems, which is compositional and fully abstract (with respect to sensible observation functions, in particular the set of reachable values for an expression), and therefore can serve as appropriate basis for semantic based analysis or manipulation of such kind of rewrite systems.

1 Introduction

Constructor based term rewriting systems (or simply *constructor systems*, CS in short) are an important subclass of term rewriting systems (TRS). The use of CS for programming has been frequently connected to the requirement of confluence. But by these days many proposals (see e.g.[13,3,10,9,11]) drop the requirement of confluence and/or termination.

On the other hand, it is widely accepted that an adequate semantics constitutes an excellent companion to any programming language. In the case of CS, an 'obvious' notion of semantics comes from defining the denotation of an expression e as the set of values reachable from e by rewriting. The notion of 'values' could be made concrete in different manners: constructor terms, outer constructor part of expressions or normal forms. Two questions arise:
– Is the semantics compositional? In our case: is the semantics of an expression determined by the semantics of its subexpressions?
– Does it capture observational equivalence? That is: for two semantically equivalent expressions e, e', is it ensured that we will *observe* the same behavior when e, e' are put in the same context? This depends on a criterion of what can be

* This work has been partially supported by the Spanish projects Merit-Forms-UCM (TIN2005-09207-C03-03), Promesas-CAM (S-0505/TIC/0407) and STAMP (TIN2008-06622-C03-01/TIN).

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 320–334, 2009.
© Springer-Verlag Berlin Heidelberg 2009

observed from an expression. In the constructor discipline point of view, one is mostly interested again in observing which constructor terms (or outer stable constructor part) can be reached by rewriting.

Somehow surprisingly, the answer to both questions is negative for the 'obvious' semantics:

Example 1. Consider the constructors a, b, c, d and the non-confluent program

$$f(c(X)) \to d(X, X) \quad choice(X, Y) \to X \quad choice(X, Y) \to Y$$

The expressions $e \equiv c(choice(a, b))$ and $e' \equiv choice(c(a), c(b))$ reach by rewriting exactly the same constructor values, namely $c(a)$ and $c(b)$. However, this does not ensure that e, e' behave the same when put in the same context. For instance, $f(e)$ can be rewritten to the constructor values $d(a, a), d(a, b), d(b, a), d(b, b)$ while $f(e')$ only to $d(a, a)$ and $d(b, b)$. More in general, this works starts by remarking that *knowing the constructor values of an expression e is not enough information to know the constructor values of $\mathcal{C}[e]$ for any given context $\mathcal{C}$*. The same example shows that the remark remains true if we replace 'constructor value' by 'normal form' or 'outer constructor part'. Using standard terminology (see Sect. 4 for definitions) all those semantics are not compositional, sound nor fully abstract.

The aim of our work can be made clear now: to define a semantics for CS that is fully abstract (compositionality and soundness will come along the way) wrt the observability criterion of reachable constructor terms.

Our starting insight is that, to recover compositionality, the semantics must not collect a *flat* set of reachable values, like is $\{c(a), c(b)\}$ for $c(choice(a, b))$, but rather a more structured and 'packaged' representation, where constructors can be applied to sets, as to reflect more appropriately the matching capabilities of expressions. In our example, and disregarding for the moment some technical details, the denotation of $c(choice(a, b))$ will be the singleton 'package' $\{c(\{a, b\})\}$, reflecting the fact that $c(choice(a, b))$ can match $c(X)$ without reducing $choice(a, b)$, while the denotation of $choice(c(a), c(b))$ will be the two-element package $\{c(\{a\}), c(\{b\})\}$. Technically, things will be a bit more complicated (see Sect. 3), in particular due to the possibility of non-termination, that will require expressing some kind of *partial* values in the semantics.

Related work. Not too much attention has been paid to the issue of semantics of TRS, at least when compared to the huge amount of research in the fields of TRS and of semantics of programming languages in general. There are nevertheless some works to be mentioned.

In [7], Boudol develops a deep theory of the space of computations of left-linear TRS and provides a computational semantics based on continuous algebras. However, his semantics still associates an expression with a flat set of (possibly infinite) values, thus presenting the problems of our Ex. 1. Moreover, [6,16] demonstrate that there are problems with achieving full abstraction for non-terminating non-deterministic systems, if the semantics is based on fixpoints and infinite (limit) values (our semantics will avoid them). In [2] a compositional semantics for conditional TRS is presented. Compositionality is understood in

a different sense, related to the issue of joining programs. In addition, the considered programs are canonical (confluent and terminating). In [1], an abstract diagnosis scheme for functional programs modeled as TRS is developed, based on some notions of semantics that again collect results of individual computations. The semantics characterization of narrowing given in [12] includes a semantics for TRS, but most of the interesting results are for confluent ones. On the other hand, the cited papers give a more general treatment of variables, which have a passive role in our paper, behaving almost as constants.

With respect to the nesting of sets inside constructor symbols, a similar idea appears in [4], to improve the efficiency of functional logic computations, in [8] as part of the design of a functional programming implementation of functional logic languages, and in [14] as a mean for programming with non-determinism in a Haskell-like ambient. All these works are much more oriented to practice, far from the aims and results of our present work. Moreover, the setting is not the same: functional logic programming for the two first (with a *call-time choice* semantics [10], having essential differences with standard rewriting) and functional programming for the last one. In [5], sets of reachable values for CS are computed; however, only topmost constructor symbols are collected and furthermore systems must be confluent and terminating.

The rest of the paper is organized as follows. Sect. 2 contains some preliminaries about TRS. Sect. 3 is the technical core of the paper, where our semantics is defined and many strong properties are proved. In Sect. 4 we discuss in detail the question of full abstraction. Finally Sect. 5 discusses potential uses of our semantics and outlines future work. Omitted proofs can be found at `http://gpd.sip.ucm.es/fraguas/papers/rta2009Long.pdf`.

2 Preliminaries

We assume a first order signature $\Sigma = DC \cup FS$, where DC and FS are two disjoint sets of *constructor* and *function* symbols resp., all them with associated arity. We write DC^n (FS^n resp.) for the set of constructor (function) symbols of arity n, and also Σ^n for any symbol of the signature of arity n. We also assume a numerable set $\mathcal{V}$ of variables. As usual notations we write $c, d, \ldots$ for constructors, $f, g, \ldots$ for functions and $X, Y, \ldots$ for variables. The set Exp of *expressions* is defined as $Exp \ni e ::= X \mid h(e_1, \ldots, e_n)$, where $X \in \mathcal{V}$, $h \in \Sigma^n$ and $e_1, \ldots, e_n \in Exp$. The set $CTerm$ of *constructed terms* (or *c-terms*) is defined like Exp, but with h restricted to DC^n (so $CTerm \subseteq Exp$).[1] We will write $e, e', \ldots$ for expressions and $t, s, \ldots$ for c-terms. The set of variables occurring in an expression e will be denoted as $var(e)$. The notation $\overline{o}$ stands for tuples of any kind of syntactic objects along the paper.

We consider also the extended signature $\Sigma_\perp = \Sigma \cup \{\perp\}$, where $\perp$ is a new 0-arity constructor symbol that stands for the undefined value. Over this signature

[1] We use the terminology Exp (for general expressions) instead of the more usual $Term$ in order to highlight the syntactic (and semantic) difference with $CTerm$ (data values).

we define the sets $Exp_\bot$ and $CTerm_\bot$ of *partial* expressions and c-terms resp. The intended meaning is that Exp and $Exp_\bot$ stand for evaluable expressions, i.e., expressions that can contain function symbols, while $CTerm$ and $CTerm_\bot$ stand for data terms representing total and partial values resp. The *shell* $|e|$ of an expression e represents its outer constructed part and is defined as: $|X| = X$; $|c(e_1, \ldots, e_n)| = c(|e_1|, \ldots, |e_n|)$; $|f(e_1, \ldots, e_n)| = \bot$. *Substitutions* $\theta \in Subst$ are mappings $\theta : \mathcal{V} \to Exp$, that extend naturally to $\theta : Exp \to Exp$.

One-hole contexts are defined as $Cntxt \ni \mathcal{C} ::= [\,] \mid h(e_1, \ldots, \mathcal{C}, \ldots, e_n)$, with $h \in \Sigma^n$. The application of a context $\mathcal{C}$ to an expression e, written by $\mathcal{C}[e]$, is defined inductively as $[\,][e] = e$ and $h(e_1, \ldots, \mathcal{C}, \ldots, e_n)[e] = h(e_1, \ldots, \mathcal{C}[e], \ldots, e_n)$.

The approximation ordering $\sqsubseteq$ is defined on expressions as the least partial ordering satisfying: *i)* $\bot \sqsubseteq e$ for all $e \in Exp_\bot$, and *ii)* $e \sqsubseteq e' \Rightarrow \mathcal{C}[e] \sqsubseteq \mathcal{C}[e']$ for all $e, e' \in Exp_\bot, \mathcal{C} \in Cntxt$.

A *constructor system* $\mathcal{P}$ (*CS*, also called *program* along this paper) is a set of rewrite rules of the form $f(\overline{t}) \to e$ where $f \in FS^n$, $e \in Exp$, $var(e) \subseteq var(\overline{t})$, and $\overline{t}$ is a linear n-tuple of c-terms, where linearity means that variables occur only once in $\overline{t}$. Given a program $\mathcal{P}$, its associated rewrite relation $\to_\mathcal{P}$ is defined as: $\mathcal{C}[l\theta] \to_\mathcal{P} \mathcal{C}[r\theta]$ for any context $\mathcal{C}$, rule $l \to r \in \mathcal{P}$ and $\theta \in Subst$. We write $\overset{*}{\to}_\mathcal{P}$ for the reflexive and transitive closure of the relation $\to_\mathcal{P}$. In the following, we will usually omit the reference to $\mathcal{P}$.

3 A Semantics for CS

In this section we present our proposed semantics, which has a logic flavor as it is based on a proof calculus. The use of proof calculi to specify the semantics of rewriting formalisms is not unfrequent. Two well-known cases correspond to the frameworks of rewriting logic [15] and CRWL [10]. We have been inspired by the philosophy of the latter, according to the following roadmap:

- We first identify the 'finite pieces' of which the denotation of expressions should be made of. These will be the *s-cterms* introduced in 3.1, capturing technically the idea of 'packaging sets below constructor' mentioned in Sect. 1.
- Then, we devise in Sect. 3.2 a proof calculus able to prove statements of the form $e \twoheadrightarrow st$ expressing that st is a finite approximation of the denotation of e. Technically, expressions will be generalized to *s-expressions*.
- The proof calculus induces a natural notion of denotation of an expression: the set of its provable approximations. Working with finite approximations makes it unnecessary to use a background of cpo's and powerdomains. This was found greatly convenient in the CRWL framework, and it is even more so in our case, where recursive nestings of constructors and sets occur. Moreover, it is known ([6,16]) that an approach based on semantic domains with infinite (limit) elements and using fixpoint techniques has technical limitations.
- If the proof calculus is designed to have a 'compositional' aspect, then one can expect compositionality of the resulting semantics, and the proof calculus is in itself a great aid to prove it. We have pursued this design principle in our proof calculus; as a result, and we have been able to prove compositionality and other relevant properties of the semantics (Sect. 3.2).

• Now, since our aim is to develop a new semantics for standard rewriting, it is essential to show that our semantics is indeed related to rewriting: this is done in Sect. 3.3 by correctness and completeness results.
• Finally, with all the previous results and an extra little effort, we are able to prove full abstraction of our semantics (Sect. 4).

3.1 SCTerms: The Pieces of the Semantics

In this section we define new syntactic notions (of expressions, cterms, etc) in order to pack different values coming from non deterministic reductions at the syntactic level, by introducing sets in the corresponding syntax. Values become *s-cterms* that must be defined in mutual recursion with *elemental s-cterms*:

$$\begin{array}{l} ESCTerm \ni est ::= X \mid c(st_1, \ldots, st_n) \\ \qquad \text{for } X \in \mathcal{V}, c \in DC^n, st_1, \ldots, st_n \in SCTerm \\ SCTerm \ni st ::= \emptyset \mid \{est_1, \ldots, est_n\} \\ \qquad \text{for } n > 0, est_1, \ldots, est_n \in ESCTerm \end{array}$$

Thus, an s-cterm is a *finite* set of elemental s-cterms, that are variables or constructors applied to s-cterms. The aim of these values is to capture the reduction of a non deterministic expression like $c(choice(a, b))$ into the single value $\{c(\{a, b\})\}$. With the same idea, but allowing also function symbols, we define *elemental s-expressions* and *s-expressions* as:

$$\begin{array}{l} ESExp \ni ese ::= X \mid h(se_1, \ldots, se_n) \\ \qquad \text{for } X \in \mathcal{V}, h \in \Sigma^n, se_1, \ldots, se_n \in SExp \\ SExp \ni se ::= \emptyset \mid \{ese_1, \ldots, ese_n\} \\ \qquad \text{for } n > 0, ese_1, \ldots, ese_n \in ESExp \end{array}$$

In the inductive definitions of $SCTerm$ and $SExp$, the base case $\emptyset$ could be hidden in the brace notation $\{est_1, \ldots, est_n\}$ just permitting $n = 0$ (in fact, we will do it sometimes). We prefer to emphasize the presence of $\emptyset$, playing the role of the undefined value (similar to $\bot$ for Exp in Sect. 2). Therefore s-cterms and s-expressions should be understood as *partial*. *Total* s-expressions and s-cterms would not use $\emptyset$, but they do not play any significant role in the following.

We can flatten an s-expression se to obtain the set $flat(se)$ of partial expressions "contained" in it: $flat(\emptyset) = \{\bot\}$ and $flat(se) = \bigcup_{ese \in se} flat(ese)$ if $se \neq \emptyset$, where the flattening of elemental s-expressions is defined as: $flat(X) = \{X\}$ and $flat(h(se_1, \ldots, se_n)) = \{h(e_1, \ldots, e_n) | e_i \in flat(se_i) \text{ for } i = 1..n\}$. Notice that $flat(se)$ is always non-empty.

The set $SSubst$ of *s-substitutions* consists of mappings $\sigma : \mathcal{V} \to SExp$ having a finite domain, where $dom(\sigma) = \{X \mid \sigma(X) \neq \{X\}\}$. Notice that s-substitutions replace variables by s-expressions (which are sets), and some care must be taken when extending s-substitutions to $ESExp$ and $SExp$:

$$\begin{array}{ll} \sigma : ESExp \to SExp & \sigma : SExp \to SExp \\ X\sigma = \sigma(X) & \{ese_1, \ldots, ese_n\}\sigma = \bigcup_{i \in \{1..n\}} ese_i\sigma \\ h(\overline{se})\sigma = \{h(\overline{se\sigma})\} & \end{array}$$

The set $SCSubst$ of *s-csubstitutions* consists of mappings $\sigma : \mathcal{V} \to SCTerm$ with a finite domain, that extend to $ESCTerm$ and $SCTerm$ analogously to the case of s-substitutions. One hole (elemental) *s-contexts* are defined as:

$$sCntxt \ni s\mathcal{C} ::= [\,] \mid \{\ldots, h(\ldots, s\mathcal{C}, \ldots), \ldots\} \quad \text{with } h \in \Sigma \text{ and } s\mathcal{C} \in sCntxt$$

The application of a context to an s-expression is defined in the natural way. Notice that s-contexts allow the hole to be only in the place of a sub-s-expression. For example, the possible s-contexts of $\{Y, c(\{X\})\}$ are $[\,]$ and $\{Y, c([\,])\}$, but not $\{[], c(\{X\})\}$ nor $\{Y, []\}$.

The preorder $\sqsubseteq$ is defined for s-expressions as the least preorder satisfying: $se \sqsubseteq se'$ if $\forall ese \in se.\exists ese' \in se'$ such that $ese \sqsubseteq ese'$, where for elemental s-expressions $\sqsubseteq$ is defined as the least preorder such that: $X \sqsubseteq X$ for any $X \in \mathcal{V}$ and $h(se_1, \ldots, se_n) \sqsubseteq h(se'_1, \ldots, se'_n)$ iff $se_i \sqsubseteq se'_i$ for $i = 1..n$. For s-substitutions, the preorder is defined as $\sigma \sqsubseteq \sigma'$ if $\sigma(X) \sqsubseteq \sigma'(X)$ for all $X \in \mathcal{V}$.

Programs remain as defined in Sect. 2. The proof calculus of the next section needs to use function rules transformed into the new syntactical framework of s-expressions. For this purpose we define the transformation of $e \in Exp$ into an s-expression $\tilde{e} \in SExp$ as: $\tilde{\bot} = \emptyset$; $\tilde{X} = \{X\}$ for any $X \in \mathcal{V}$; $\widetilde{h(e_1, \ldots, e_n)} = \{h(\tilde{e_1}, \ldots, \tilde{e_n})\}$, with $h \in \Sigma^n$. The transformation $\tilde{\mathcal{C}}$ of a context $\mathcal{C}$ is defined in the natural way, so that it verifies $\widetilde{\mathcal{C}[e]} = \tilde{\mathcal{C}}[\tilde{e}]$. On the other hand, $\tilde{\sigma}$ is defined as $\tilde{\sigma}(X) = \widetilde{\sigma(X)}$, for $\sigma \in Subst$.

3.2 A Proof Calculus

Our goal in this section is to devise a proof calculus that specifies which $SCTerm$s correspond to a given expression under a given CS. Inspired by the *CRWL* proof calculus of [10], to achieve a compositional aspect of the calculus expressions are evaluated in an innermost way, and the use of any transitivity rule is avoided. By the use of partial s-cterms as values, the 'compositional' innermost procedure of the calculus does not enforce strictness of functions, which is essential to achieve completeness of our semantics wrt term rewriting even for non-terminating CS.

Besides, during parameter passing the variables in the program rules will be instanciated with partial s-cterms. Then it is possible to end up evaluating expressions with some $SCTerm$ "inside" (as a subexpression), even when starting the computation from an ordinary $e \in Exp$. So, instead of dealing only with expressions from Exp, our calculus will compute the partial s-cterms corresponding to any given partial s-expression. Finally, the mapping $\tilde{\ }$ will be used in combination with our logic to get the $SCTerm$s corresponding to a given Exp.

To be precise, our proof calculus will prove reduction statements of the form $se \twoheadrightarrow st$ with $se \in SExp$ and $st \in SCTerm$, expressing that st represents a finite approximation to one of the possible structured sets of values for se.

The calculus is presented in Fig. 1. Rule E (empty) allows us to avoid the evaluation of any expression, in order to get a non-strict semantics. Rules RR (restricted reflexivity) and DC (decomposition) work with singleton sets and allow us to reduce any variable to itself, and to decompose the evaluation of

$$\textbf{(E)}\quad se \twoheadrightarrow \emptyset \qquad\qquad \textbf{(RR)}\quad \{X\} \twoheadrightarrow \{X\} \quad \text{if } X \in \mathcal{V}$$

$$\textbf{(DC)}\quad \frac{se_1 \twoheadrightarrow st_1 \;\ldots\; se_n \twoheadrightarrow st_n}{\{c(se_1,\ldots,se_n)\} \twoheadrightarrow \{c(st_1,\ldots,st_n)\}} \quad \text{if } c \in CS$$

$$\textbf{(More)}\quad \frac{se \twoheadrightarrow st_1 \;\ldots se \twoheadrightarrow st_n}{se \twoheadrightarrow st_1 \cup \ldots \cup st_n}$$

$$\textbf{(Less)}\quad \frac{\{esa_1\} \twoheadrightarrow st_1 \;\ldots\; \{esa_m\} \twoheadrightarrow st_m}{\{ese_1,\ldots,ese_n\} \twoheadrightarrow st_1 \cup \ldots \cup st_m} \quad \begin{array}{l}\text{if } n \geq 2, m > 0, \text{ for any} \\ \{esa_1,\ldots,esa_m\} \subseteq \{ese_1,\ldots,ese_n\}\end{array}$$

$$\textbf{(ROR)}\quad \frac{se_1 \twoheadrightarrow \tilde{p_1}\theta \;\ldots\; se_n \twoheadrightarrow \tilde{p_n}\theta \;\; \tilde{r}\theta \twoheadrightarrow st}{\{f(se_1,\ldots,se_n)\} \twoheadrightarrow st} \quad \begin{array}{l}\text{if } (f(p_1,\ldots,p_n) \to r) \in \mathcal{P} \\ \theta \in SCSubst\end{array}$$

Fig. 1. A proof calculus for constructor systems

a constructor-rooted elemental s-expression. Rule More allows us to compute more than one value for an s-expression, and to collect these values together. Rule Less allows us to discard some elemental s-expressions from the s-expression under evaluation. Finally rule ROR (run-time[2] outer reduction) expresses that to evaluate a function call we must first evaluate its arguments to get an instance of a program rule, perform parameter passing (by means of an $SCSubst\ \theta$) and then reduce the instantiated right-hand side. The use of $SCSubst$s is fundamental to get the exact behaviour of term rewriting, because then the branching information associated to the computation of each $\tilde{p_i}\theta$ is not lost in some kind of flattening to a set of c-terms, but kept into the structured representation of $SCTerm$s.

We write $\mathcal{P} \vdash se \twoheadrightarrow st$ to express that $se \twoheadrightarrow st$ is derivable in our calculus under the CS $\mathcal{P}$. The *denotation* of an s-expression se under $\mathcal{P}$ is defined as $[\![se]\!]^{\mathcal{P}} = \{st \in SCTerm \mid \mathcal{P} \vdash se \twoheadrightarrow st\}$. In the following we will usually omit the reference to $\mathcal{P}$.

Example 2. Consider the CS of Ex. 1. We can use our calculus to prove the statement $f(c(\widetilde{choice}(a,b))) \twoheadrightarrow \widetilde{d(a,b)}$ (some steps have been omitted for the sake of conciseness, and *choice* is abbreviated to *ch*):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{a\} \twoheadrightarrow \{a\}}\,\text{DC} \quad \dfrac{}{\{b\} \twoheadrightarrow \emptyset}\,\text{E} \quad \dfrac{\ldots}{\{a\} \twoheadrightarrow \{a\}}}{\{ch(\{a\},\{b\})\} \twoheadrightarrow \{a\}}\,\text{ROR} \quad \dfrac{\ldots}{\{ch(\{a\},\{b\})\} \twoheadrightarrow \{b\}}\,\text{ROR}}{\{ch(\{a\},\{b\})\} \twoheadrightarrow \{a,b\}}\,\text{More} \quad (*)}{\{c(\{ch(\{a\},\{b\})\})\} \twoheadrightarrow \{c(\{a,b\})\}}\,\text{DC} \quad \dfrac{}{\{d(\{a,b\},\{a,b\})\} \twoheadrightarrow \{d(\{a\},\{b\})\}}\,\text{DC}}{f(c(\widetilde{ch(a,b)})) \equiv \{f(\{c(\{ch(\{a\},\{b\})\})\})\} \twoheadrightarrow \{d(\{a\},\{b\})\} \equiv \widetilde{d(a,b)}}\,\text{ROR}$$

where $(*)$ is the derivation:

$$\dfrac{\dfrac{\dfrac{}{\{a\} \twoheadrightarrow \{a\}}\,\text{DC}}{\{a,b\} \twoheadrightarrow \{a\}}\,\text{Less} \quad \dfrac{\ldots}{\{a,b\} \twoheadrightarrow \{b\}}}{\{d(\{a,b\},\{a,b\})\} \twoheadrightarrow \{d(\{a\},\{b\})\}}\,\text{DC}$$

[2] The prefix 'run-time' comes from 'run-time choice', a term often applied ([13,10]) to the parameter passing mechanism of term rewriting.

On the other hand, $\widetilde{d(a,b)}$ is not a correct value for $f(\widetilde{choice(\widetilde{c(a)}, c(b))})$, because in that expression the evaluation of $\widetilde{choice(\widetilde{c(a)}, c(b))}$ has to be performed in order to get a value matching the argument of the left-hand side of the only rule for f, and the only matching values for it are $\widetilde{c(a)}$, $\widetilde{c(b)}$ and $\{c(\emptyset)\}$, as for example $\{c(\{a\}), c(\{b\})\}$ does not match $\widetilde{c(X)}$.

Notice that, structurally, a denotation $[\![se]\!]$ is a possibly infinite set of s-cterms, Infinite denotations might appear with non-terminating programs. Notice, however, that the elements are s-cterms that are, by construction, finite objects. Thus, we avoid the presence of infinite values as elements of denotations, escaping from the problems mentioned in [6,16].

As we anticipated above, even when proving a reduction $\tilde{e} \twoheadrightarrow \tilde{t}$ for $e \in Exp$ and $t \in CTerm$, we may find premises of the shape $se \twoheadrightarrow st$ for $se \in SExp, st \in SCTerm$, because the substitutions used for parameter passing in ROR may introduce sets in $\tilde{r}\theta$, as we can see in the second premise of the first application of ROR, in Ex. 2. But in fact the kind of s-expressions that we may find in the proof for some reduction for an expression is more restricted. It is easy to prove that in any proof for any statement $\tilde{e} \twoheadrightarrow st$ with $e \in Exp$ we have that in any premise $se' \twoheadrightarrow st'$ for it, $se \in trSExp$, a set defined as $trSExp \ni tr ::= st \mid \{h(tr_1, \ldots, tr_n)\}$, with $st \in SCTerm, h \in \Sigma, tr_1, \ldots, tr_n \in trSExp$.
This suggests that we could have defined our logic to prove reductions $se \twoheadrightarrow st$ with $se \in trSExp$ and $st \in SCTerm$ only, but we think that it is more profitable to define it to deal with the more general case of $se \in SExp$. First of all, then we get a logic that can handle a more general kind of syntactic objects, and therefore that could be used to express other formalism apart from term rewriting. We could slightly modify the rule ROR to accept not only CSs but in general *"s-expression rewriting systems" (sCSs)*, consisting of rules $\{f(st_1, \ldots, st_n)\} \twoheadrightarrow se$. This way the original formulation of ROR becomes a particular case of the new version, that works with CSs adapted to sCSs by means of $\tilde{_}$. This would be similar to what is done in [17] to express term rewriting, term graph rewriting and noncopying rewriting by means of the more general framework of marked term rewriting. We consider this an interesting possible subject of future work. On the other hand, working with reductions of s-expressions allows us to formulate more general and powerful results about the semantics, like the *polarity* property stated below, which become easier to prove because of their generality (that provides stronger induction hypotheses), and that can be then easily applied to the more restricted case. In all the results of this and the next section we assume a given CS and omit mentioning it.

Proposition 1 (Polarity). *Let $se, se' \in SExp$, $st, st' \in SCTerm$. If $se \sqsubseteq se'$ and $st' \sqsubseteq st$ then $st \in [\![se]\!]$ implies $st' \in [\![se']\!]$.*

Our semantics also enjoys the following monotonicity property related to s-substitutions. It is formulated for the preorder $\sqsubseteq$ and also for the preorder $\trianglelefteq$ over $SSubst$, defined by $\sigma \trianglelefteq \sigma'$ iff $\forall X \in \mathcal{V}, [\![\sigma(X)]\!] \subseteq [\![\sigma'(X)]\!]$.

Proposition 2 (Monotonicity of substitutions). *Let* $se \in SExp$, $\sigma, \sigma' \in SSubst$. *If* $\sigma \trianglelefteq \sigma'$ *or* $\sigma \sqsubseteq \sigma'$ *then* $[\![se\sigma]\!] \subseteq [\![se\sigma']\!]$.

One of the most important properties of our logic is its *compositionality*, a property very close to the DET-additivity property for algebraic specifications of [13], which will be one of the keys for full abstraction.

Theorem 1 (Compositionality). *For all* $s\mathcal{C} \in sCntxt, se \in SExp$,

$$[\![s\mathcal{C}[se]]\!] = \bigcup_{st \in [\![se]\!]} [\![s\mathcal{C}[st]]\!]$$

As a consequence: $[\![se]\!] = [\![se']\!] \Leftrightarrow \forall s\mathcal{C}.\ [\![s\mathcal{C}[se]]\!] = [\![s\mathcal{C}[se']]\!]$.

Regarding *closedness* under susbtitutions, as we use $SCSubst$ for parameter passing it is natural to have closedness of reductions under this type of substitutions. Besides, as rewriting is closed under $Subst$ it is expected to have some kind of closedness for $Subst$ too. But in general it is not true that for any $st \in SCTerm, \sigma \in SSubst$ we have $st\sigma \in SCTerm$, therefore it makes no sense to expect that $se \twoheadrightarrow st$ implies $se\sigma \twoheadrightarrow st\sigma$, as the reductions in our logic are from $SExp$ to $SCTerm$. Nevertheless we still can say something about that, as we can see in the following property.

Proposition 3 (Closedness under substitutions). *Let* $se \in SExp$ *and* $st \in [\![se]\!]$. *Then: a)* $\forall \theta \in SCSubst, st\theta \in [\![se\theta]\!]$. *b)* $\forall \sigma \in SSubst, [\![st\sigma]\!] \subseteq [\![se\sigma]\!]$.

All these properties are powerful tools to reason about the denotation of s-expressions. And this reasoning power is transferred to the term rewriting universe through the adequacy results that we will see in the next section, thus opening paths for the development of new reasoning techniques for CSs.

3.3 Relation with Rewriting

The nice properties of our logic will be useless unless they are accompanied by strong adequacy results with respect to the term rewriting relation. We first address the *completeness* of our logic, i.e., that the semantics of any expression captures any c-term reachable from it by rewriting. As a first result we have:

Proposition 4. *For all* $e, e' \in Exp$, *if* $e \rightarrow^* e'$ *then* $[\![\tilde{e}']\!] \subseteq [\![\tilde{e}]\!]$.

The keys for its proof are Th. 1 and the following Lemma 1, expressing that any reduction $se\sigma \twoheadrightarrow st$ needs to use only a finite amount of the information contained in σ, formalized through the notion of denotation of an $SSubst$, defined as $[\![\sigma]\!] = \{\theta \in SCSubst \mid \forall X \in \mathcal{V}, \sigma(X) \twoheadrightarrow \theta(X)\}$.

Lemma 1. *Let* $\sigma \in SSubst, se \in SExp, st \in SCTerm$. *If* $se\sigma \twoheadrightarrow st$ *then there exists* $\theta \in [\![\sigma]\!]$ *such that* $se\theta \twoheadrightarrow st$.

Now we can apply Prop. 4 to get the following strong completeness result.

Theorem 2 (Completeness). *For all* $e, e' \in Exp, t \in CTerm$,
a) $e \rightarrow^* e'$ *implies* $|\widetilde{e'}| \in [\![\tilde{e}]\!]$ *b)* $e \rightarrow^* t$ *implies* $\widetilde{t} \in [\![\tilde{e}]\!]$

$_ \vdash _ \lessdot _ \subseteq CS \times SCTerm \times Exp$	$_ \vdash _ \lessdot _ \subseteq CS \times ESCTerm \times Exp$
$\mathcal{P} \vdash st \lessdot e$ if $\forall est \in st, \mathcal{P} \vdash est \lessdot e$	$\mathcal{P} \vdash X \lessdot e$ if $\mathcal{P} \vdash e \to^* X$
	$\mathcal{P} \vdash c(\overline{st}) \lessdot e$ if $\mathcal{P} \vdash e \to^* c(\overline{e})$ for some $\overline{e}$ such that $\forall e_i \in \overline{e}, \mathcal{P} \vdash st_i \lessdot e_i$

Fig. 2. Domination relation

Proof. For *a)*, it is easy to prove that $\forall e \in Exp, \tilde{e} \twoheadrightarrow \widetilde{|e|}$, by induction on the structure of e. But then $\widetilde{|e'|} \in [\![\tilde{e'}]\!] \subseteq [\![\tilde{e}]\!]$ by Prop. 4. For *b)*, just notice that $\forall t \in CTerm, |t| \equiv t$, and so $\tilde{e} \twoheadrightarrow \widetilde{|t|} \equiv \tilde{t}$, by *a)*.

We also want our logic to be *correct*, in the sense that the semantics of any expression does not compute more c-terms than those reachable by rewriting. One key ingredient will be the *domination relation* $_ \lessdot _$ defined in Fig. 2 (we will omit the prefix "$\mathcal{P} \vdash$" when deducible from the context). With this relation we try to transfer to the rewriting world the finer distinction between sets of values that the structured representation of $SCTerm$ allows us to perform. This way under the CS of Ex. 1 we have $\{c(\{a, b\})\} \lessdot c(choice(a, b))$ but not $\{c(\{a, b\})\} \lessdot choice(c(a), c(b))$. The domination relation $_ \lessdot _$ has a strong relation with our semantics, as stated in the following result:

Lemma 2 (Domination). *For all $e \in Exp, st \in SCTerm$: $st \in [\![\tilde{e}]\!]$ iff $st \lessdot e$.*

Notice that $_ \lessdot _$ only talks about reductions for $\tilde{e}$ with $e \in Exp$, and so it cannot be used to formulate properties like those seen in Sect. 3.2, although it inherits them through Lemma 2. But he good thing about $_ \lessdot _$ is that it already has a strong connection with rewriting, as it is defined by means of rewriting derivations. Hence we can perform a simple induction on the structure of $SCTerm$ and $ESCTerm$ to prove the following result, which uses the notion of flattening defined in Sect. 3.1.

Lemma 3. *Let $st \in SCTerm$, $est \in ESCTerm$, $e \in Exp$, and assume $t \in flat(st)$. If $st \lessdot e$ then $e \to^* e'$ for some $e' \in Exp$ such that $t \sqsubseteq |e'|$.*

And now we are ready to state and prove our main correctness result.

Theorem 3 (Correctness). *Let $e \in Exp, st \in SCTerm, t \in CTerm_\perp$:*

a) If $st \in [\![\tilde{e}]\!]$ and $t \in flat(st)$, then $e \to^* e'$ for some $e' \in Exp$ such that $t \sqsubseteq |e'|$.
b) If $\tilde{t} \in [\![\tilde{e}]\!]$, then $e \to^* e'$ for some $e' \in Exp$ such that $t \sqsubseteq |e'|$.
c) Besides, in a) or b), if $t \in CTerm$, then $e \to^* t$.

Proof. We get *a)* just chaining Lemma 2 and Lemma 3. Concerning *b)*, we can prove that $\forall t \in CTerm_\perp, flat(\tilde{t}) = \{t\}$ by induction on $CTerm_\perp$, and chain it with *a)*. Finally *c)* is a consequence of *a)* and *b)*, because if $t \in CTerm$ then it is maximal wrt $\sqsubseteq$, hence $t \sqsubseteq |e'|$ implies $t \equiv |e'|$. But that implies there is no $\perp$ in $|e'|$, therefore $e' \in CTerm$ and $e' \equiv |e'| \equiv t$, and so $e \to^* e' \equiv t$.

Correctness is the point where left linearity of programs is essential. Without left linearity, the results of Sect. 3.2 still hold, but the semantics becomes incorrect wrt rewriting. For instance, if $\mathcal{P} \equiv \{f(X, X) \to a\}$ and $e \equiv f(a, b)$ then it can be shown that $\tilde{a} \in [\![\tilde{e}]\!]$ but $e \not\to^* a$, contradicting Th. 3.

4 Full Abstraction

The semantics $[\![se]\!]^{\mathcal{P}}$ of Sect. 3 is defined for s-expressions, but induces naturally a notion of semantics for ordinary expressions $e \in Exp$:

$$[\![e]\!]^{\mathcal{P}}_S = [\![\tilde{e}]\!]^{\mathcal{P}}(= \{st \in SCTerm | \tilde{e} \twoheadrightarrow st\})$$

In this section we discuss full abstraction in the context of CS and show that $[\![_]\!]_S$ achieves it, in contrast to semantics directly based on sets of results, informally described in Sect. 1. In a fully abstract semantics, two expressions have the same semantics if and only if they are observationally indistinguishable. For this to be meaningful, one must choose a criterion of observability. The problem of full abstraction was first investigated by Plotkin [18] for PCF (a simple functional programming language), and since then is a standard topic when dealing with program semantics (see e.g. [19]). It is common to adjust its definition to the characteristics of the language under consideration. In the context of functional-like programming languages like PCF the condition for full abstraction is usually stated as:

$$[\![e]\!] = [\![e']\!] \Leftrightarrow \mathcal{O}(\mathcal{C}[e]) = \mathcal{O}(\mathcal{C}[e']), \text{for any context } \mathcal{C} \tag{1}$$

where $\mathcal{O}$ is the observation function of interest. Programs do not need to be mentioned, because programs and expressions can be identified by contemplating the evaluation of e under $\mathcal{P}$ as the evaluation of a large λ-expression or *let*-expression embodying $\mathcal{P}$ and e. Contexts pose no problems either. In our case, since programs (CS) are kept different from expressions, some care must be taken. It might happen that $\mathcal{P}$ has not enough syntactical elements and rules to build interesting distinguishing contexts. For instance, if in Ex. 1 we drop the definition of f, then we cannot built a context that distinguishes $c(choice(a, b))$ from $choice(c(a), c(b))$. This would imply that soundness or full abstraction would not be intrinsic to the semantics, but would depend on the program. What we need is requiring the right part of (1) to hold for all contexts that might be obtained by extending $\mathcal{P}$ with new auxiliary functions. To be more precise, we say that $\mathcal{P}'$ is a *safe extension* of $(\mathcal{P}, e)$ if $\mathcal{P}' = \mathcal{P} \cup \mathcal{P}''$, where $\mathcal{P}''$ does not include defining rules for any function symbol occurring in $\mathcal{P}$ or e. Any sensible notion of semantics should verify $[\![e]\!]^{\mathcal{P}} = [\![e]\!]^{\mathcal{P}'}$ when $\mathcal{P}'$ safely extends (P, e). This happens indeed for all the semantics considered below.

Things are now prepared to give the following definition:

Definition 1 (Observations, full abstraction).
(a) A semantic function (a semantics*, in short) is a function $[\![_]\!]$ associating a semantic value (taken from a set $\mathcal{D}$) to each expression e under a given program $\mathcal{P}$. We write $[\![e]\!]^{\mathcal{P}}$ for such value.*
(b) An observation function *is a function $\mathcal{O}$ associating a set of observation values (or* observables*, taken from a set Obs) to each expression e under a given program $\mathcal{P}$. We write $\mathcal{O}^{\mathcal{P}}(e)$ for it.*
(c) A semantics is fully abstract *wrt $\mathcal{O}$ iff for any $\mathcal{P}$ and $e, e' \in Expr$, the following two conditions are equivalent:*

(i) $[\![e]\!]^{\mathcal{P}} = [\![e']\!]^{\mathcal{P}}$ *(ii)* $\mathcal{O}^{\mathcal{P}'}(\mathcal{C}[e]) = \mathcal{O}^{\mathcal{P}'}(\mathcal{C}[e'])$ *for any* $\mathcal{P}'$ *safely extending* $(\mathcal{P}, e), (\mathcal{P}, e')$ *and any* $\mathcal{C}$ *built with the signature of* $\mathcal{P}'$.
In words: semantic equality is equivalent to observational indistinguishability.
(d) A notion weaker than full abstraction is: a semantics is sound *wrt* $\mathcal{O}$ *iff the condition* (i) *above implies the condition* (ii).
In words: semantic equality implies observational indistinguishability.
(e) A semantics is compositional *iff for any* $\mathcal{P}$ *and* $e, e' \in Expr$, *the following two conditions are equivalent:*
(i) $[\![e]\!]^{\mathcal{P}} = [\![e']\!]^{\mathcal{P}}$ *(ii)* $[\![\mathcal{C}[e]]\!]^{\mathcal{P}} = [\![\mathcal{C}[e']]\!]^{\mathcal{P}}$ *for any* $\mathcal{C}$ *built with the signature of* $\mathcal{P}$.
In words: the semantics of an expression depends only on the semantics of its subexpressions. Notice that $(ii) \Rightarrow (i)$ *holds trivially (take* $\mathcal{C} = [\ ]$*).*

In the next definition we collect some notions of semantics and observables for the case of CS. Our new contributed semantics are $[\![_]\!]_S$ and $[\![_]\!]_{S'}$; the rest are the 'obvious' semantics of Sect. 1. As usual, we omit the program $\mathcal{P}$ in notations.

Definition 2 (Semantics and observations for CSs).
We consider the following semantics for expressions $e \in Exp$:

$$[\![e]\!]_{rw} = \{e' \mid e \to^* e'\} \qquad [\![e]\!]_{nf} = \{e' \mid e \to^* e', e' \text{in normal form}\}$$
$$[\![e]\!]_t = \{t \in CTerm \mid e \to^* t\} \qquad [\![e]\!]_{t_\perp} = \{t \in CTerm_\perp \mid \exists e'.(e \to^* e' \wedge t \sqsubseteq |e'|)\}$$
$$[\![e]\!]_S = [\![\tilde{e}]\!] \qquad [\![e]\!]_{S'} = \bigcup_{st \in [\![e]\!]_S} st.$$

and two observation functions for expressions: $\mathcal{O}_t(e) = [\![e]\!]_t$ *and* $\mathcal{O}_{t_\perp}(e) = [\![e]\!]_{t_\perp}$.

Some remarks:

- It is clear that $[\![e]\!]_t \subseteq [\![e]\!]_{nf} \subseteq [\![e]\!]_{rw}$, and also $[\![e]\!]_t \subseteq [\![e]\!]_{t_\perp}$.
- Notice that some of the sets above can play at the same time the role of semantic values and of observation values.
- $[\![e]\!]_S$ was introduced earlier in this section. $[\![e]\!]_{S'}$ is a simplified variant, making more readable the semantics of expressions, because $[\![e]\!]_S$ is a set of finite sets of *est*s, while $[\![e]\!]_{S'}$ is simply a set of *est*s. However $[\![_]\!]_S$ has been technically more convenient for proving properties of the semantics, due to its more direct connection to the proof calculus. Both semantics are essentially the same:

Proposition 5. *For any* $e, e' \in Exp$, $[\![e]\!]_S = [\![e']\!]_S \Leftrightarrow [\![e]\!]_{S'} = [\![e']\!]_{S'}$

The next result shows that, although $\mathcal{O}_t, \mathcal{O}_{t_\perp}$ define different observations, it is irrelevant which is chosen, as far as full abstraction is concerned.

Proposition 6. *Assume a given semantics* $[\![_]\!]$. *Then:*

$$[\![_]\!] \text{ is fully abstract wrt } \mathcal{O}_t \Leftrightarrow [\![_]\!] \text{ is fully abstract wrt } \mathcal{O}_{t_\perp}.$$

Now we show that the first four semantics, $[\![]\!]_{rw}_, [\![_]\!]_{nf}, [\![_]\!]_t, [\![_]\!]_{t_\perp}$ do not have good properties, as was informally discussed in Sect. 1. We use $\mathcal{O}_t$ in the result but, according to the previous result, $\mathcal{O}_{t_\perp}$ could be used instead.

Proposition 7.

(a) $[\![_]\!]_{rw}, [\![_]\!]_{nf}, [\![_]\!]_t, [\![_]\!]_{t_\perp}$ *are not fully abstract wrt* $\mathcal{O}_t$.
(b) Moreover, $[\![_]\!]_{nf}, [\![_]\!]_t, [\![_]\!]_{t_\perp}$ *are not compositional, nor sound wrt* $\mathcal{O}_t$.
(c) $[\![_]\!]_{nf}$ ($[\![_]\!]_t$ *resp.) remains not compositional nor sound wrt* $\mathcal{O}_t$ *even if programs are restricted to be confluent (confluent and terminating, resp.).*

Proof. (a) For $[\![_]\!]_{nf}, [\![_]\!]_t, [\![_]\!]_{t_\perp}$, (a) is implied by (b). For $[\![_]\!]_{rw}$, just add a new function $g(X) \to f(X)$ to Ex. 1. It is easy to see that $f(a)$ and $g(a)$ are contextually indistinguishable wrt $\mathcal{O}_t$, but $[\![f(a)]\!]_{rw} \neq [\![g(a)]\!]_{rw}$.
(b) Example 1 of Sect. 1 serves for all the three semantics.
(c) For $[\![_]\!]_{nf}$, consider the program $\{f \to f, g \to c(f), h(c(X)) \to a\}$. We have $[\![f]\!]_{nf} = [\![g]\!]_{nf} = \emptyset$, but $[\![h(f)]\!]_{nf} = \emptyset \neq \{a\} = [\![h(g)]\!]_{nf}$, proving at the same time not compositionality and unsoundness. For $[\![_]\!]_t$, replace the above program by $\{f \to h(a), g \to c(f), h(c(X)) \to a\}$.

Finally, we show that our semantics do not present those problems.

Theorem 4 (Compositionality and full abstraction of $[\![_]\!]_S$).

$[\![_]\!]_S$ *and* $[\![_]\!]_{S'}$ *are compositional and fully abstract wrt* $\mathcal{O}_t$ *and* $\mathcal{O}_{t_\perp}$.

Proof. We prove the results for $[\![_]\!]_S$ and $\mathcal{O}_t$. For $[\![_]\!]_{S'}$ and $\mathcal{O}_{t_\perp}$, just use propositions 5 and 6. Compositionality follows from definition of $[\![_]\!]_S$ and compositionality of $[\![_]\!]$ (Th. 1).
For full abstraction, let $\mathcal{P}$ be any CS, and $e, e' \in Expr$. We must prove:

$$[\![e]\!]^{\mathcal{P}}_S = [\![e']\!]^{\mathcal{P}}_S \Leftrightarrow \forall \mathcal{P}', \mathcal{C} \ . \mathcal{O}^{\mathcal{P}'}_t(\mathcal{C}[e]) = \mathcal{O}^{\mathcal{P}'}_t(\mathcal{C}[e'])$$

where $\mathcal{P}'$ ranges over safe extensions of $(\mathcal{P}, e)$ and $(\mathcal{P}, e')$, and $\mathcal{C}$ over contexts built with the signature of $\mathcal{P}'$.
$\boxed{\Rightarrow}$ Assume $[\![e]\!]^{\mathcal{P}}_S = [\![e']\!]^{\mathcal{P}}_S$. Since $\mathcal{P}'$ is a safe extension, we know that $[\![e]\!]^{\mathcal{P}'}_S = [\![e']\!]^{\mathcal{P}'}_S$. We prove $\mathcal{O}^{\mathcal{P}'}_t(\mathcal{C}[e]) \subseteq \mathcal{O}^{\mathcal{P}'}_t(\mathcal{C}[e'])$ (the other inclusion is similar). Let $t \in \mathcal{O}^{\mathcal{P}'}_t(\mathcal{C}[e])$, which means $\mathcal{C}[e] \to^*_{\mathcal{P}'} t$. By Th. 2, $\tilde{t} \in [\![\mathcal{C}[e]]\!]^{\mathcal{P}'}_S = [\![\mathcal{C}[e']]\!]^{\mathcal{P}'}_S$, where the last equality is justified by compositionality. Then, since $t \in flat(\tilde{t})$, we conclude from Th. 3 that $\mathcal{C}[e'] \to^*_{\mathcal{P}'} t$, that is, $t \in \mathcal{O}^{\mathcal{P}'}_t(\mathcal{C}[e'])$, as desired.

For the other implication we need two auxiliary constructions enabling to build a context that distinguishes two expressions having different semantics:

- Given $st \in SCTerm$, we define a c-term $\widehat{st}$ 'mirroring' st as follows:

$$\widehat{\emptyset} = \langle\,\rangle_0 \qquad \widehat{\{X\}} = X \ (X \in \mathcal{V}) \qquad \widehat{\{c(\overline{st_i})\}} = c(\overline{\widehat{st_i}})$$
$$\widehat{\{est_1, \ldots, est_n\}} = \langle \widehat{\{est_1\}}, \ldots, \widehat{\{est_n\}} \rangle_n \ (n > 1)$$

where $\langle_\rangle_n$ $(n \geq 0)$ are new tuple-forming constructor symbols. It is assumed here that $SCTerm, ESCTerm$ are equipped with any standard ordering.

- Given $st \in SCTerm$, the program $\mathcal{P}_{st}$ defines new functions $f_{st}, \ldots$ as follows:

$$f_\emptyset(X) \to \langle\,\rangle_0 \qquad f_{\{X\}}(U) \to U \ (X \in \mathcal{V}) \qquad f_{\{c(\overline{st_i})\}}(c(\overline{U_i})) \to c(\overline{f_{st_i}(U_i)})$$
$$f_{\{est_1, \ldots, est_n\}}(U) \to \langle f_{\{est_1\}}(U), \ldots, f_{\{est_n\}}(U) \rangle_n \ (n > 1)$$

The roles of $\widehat{st}, \mathcal{P}_{st}$ are made clear by the following lemma:

Lemma 4. *For any* $\mathcal{P}, st, e$: $st \in [\![e]\!]^{\mathcal{P}}_S \Leftrightarrow f_{st}(e) \to^*_{\mathcal{P}'} \widehat{st}$, *where* $\mathcal{P}' \equiv \mathcal{P} \cup \mathcal{P}_{st}$.

We can now proceed with the proof of the pending implication.
$\boxed{\Leftarrow}$ Assume that $\mathcal{O}^{\mathcal{P}'}_t(\mathcal{C}[e]) = \mathcal{O}^{\mathcal{P}'}_t(\mathcal{C}[e'])$ for any safe extension $\mathcal{P}'$ and context $\mathcal{C}$. We prove $[\![e]\!]^{\mathcal{P}}_S \subseteq [\![e']\!]^{\mathcal{P}}_S$ (the other inclusion is similar). Let $st \in [\![e]\!]^{\mathcal{P}}_S$. Let $\mathcal{P}' \equiv \mathcal{P} \cup \mathcal{P}_{st}$, which is a safe extension of $(P, e), (P, e')$. Lemma 4 ensures $f_{st}(e) \to^*_{\mathcal{P}'} \widehat{st}$, which means $\widehat{st} \in \mathcal{O}^{\mathcal{P}'}_t(f_{st}(e))$. Now, since $\mathcal{P}'$ is a safe extension, observational equivalence of e, e' implies $\widehat{st} \in \mathcal{O}^{\mathcal{P}'}_t(f_{st}(e'))$, which means $f_{st}(e') \to^*_{\mathcal{P}'} \widehat{st}$. Again by Lemma 4, we conclude that $st \in [\![e']\!]^{\mathcal{P}}_S$, as desired.

5 Conclusions

In this paper we have provided a semantics for constructor systems that is fully abstract with respect to natural notions of observation that extract the outer constructor part of outcomes as relevant information of computations. To the best of our knowledge, this is the first time that full abstraction has been achieved for this class of programs and observations. Along the way to this result we have made some contributions: after noticing that 'obvious' semantics directly based on rewrite sequences lack compositionality, our main insight has been that it can be recovered by recursively packaging sets of results below constructor symbols. That insight has been realized at the technical level by introducing s-cterms as suitable semantic values, and giving a proof calculus able to derive reachable s-cterms from a given expression. Previous to full abstraction, we have proved a bunch of good properties of the semantics: polarity, compositionality, closedness under substitutions, correctness and completeness with respect to rewriting.

We expect our semantics to be a useful tool for CS-based program manipulation. We remark that, for instance, to justify the correctness of a CS-transformation by proving preservation of reachable values could be incorrect if transformations are to be used locally. Our semantics could be a better option, as illustrated by the following simple example: consider a program piece made of the rules $f \to c(g),\ g \to e,\ g \to e'$, where e, e' are expressions that costly reduce to a, b respectively. A 'obvious' partial evaluation might suggest replacing f's definition by the optimized one $f \to c(a),\ f \to c(b)$, leading to a transformed program $\mathcal{P}'$, presumably equivalent to $\mathcal{P}$. This is wrong: if $\mathcal{P}$ defines also h by the rule $h(c(X)) \to d(X, X)$, then $h(f)$ behaves different with both definitions of f. This is detected in our semantics (using e.g. the variant $[\![_]\!]_{S'}$ of Def. 2), because $[\![f]\!]^{\mathcal{P}}_{S'} \ni c(\{a, b\}) \notin [\![f]\!]^{\mathcal{P}'}_{S'}$. Imagine, however, that the original program piece was $f \to g,\ g \to c(e),\ g \to c(e')$. In this case, the 'obvious' partial evaluation of f would lead again to $f \to c(a),\ f \to c(b)$. Is this right now? Yes, because $[\![f]\!]^{\mathcal{P}}_{S'} = [\![f]\!]^{\mathcal{P}'}_{S'}$, and full abstraction of $[\![_]\!]_{S'}$ makes the rest. A deeper investigation of these issues is planned for the future.

There are other aspects not yet accomplished that can be subject of future work. We plan to investigate full abstraction for other notions of observations, in two different senses: by giving a more active role to variables (as happens in [2,1,12]) taking into account that, for instance, variables can be subject of narrowing substitutions; and by replacing our 'may-convergent' or 'angelic' view of non-determinism (in which two expressions may have the same

semantics even if one admits divergent reductions while the other does not) by a 'must-convergence' view where divergence plays a role. Dropping the constructor restriction is also interesting, replacing the role of constructor values by appropriate alternatives. Finally, incorporating s-expressions to the syntax of programs, as discussed in Sect. 3.2, could lead to more expressive programs.

References

1. Alpuente, M., Comini, M., Escobar, S., Falaschi, M., Lucas, S.: Abstract diagnosis of functional programs. In: Leuschel, M.A. (ed.) LOPSTR 2002. LNCS, vol. 2664, pp. 1–16. Springer, Heidelberg (2003)
2. Alpuente, M., Falaschi, M., Ramis, M., Vidal, G.: Narrowing approximations as an optimization for equational logic programs. In: Penjam, J., Bruynooghe, M. (eds.) PLILP 1993. LNCS, vol. 714, pp. 391–409. Springer, Heidelberg (1993)
3. Antoy, S.: Optimal non-deterministic functional logic computations. In: Hanus, M., Heering, J., Meinke, K. (eds.) ALP 1997 and HOA 1997. LNCS, vol. 1298, pp. 16–30. Springer, Heidelberg (1997)
4. Antoy, S., Iranzo, P.J., Massey, B.: Improving the efficiency of non-deterministic computations. ENTCS, 64 (2002)
5. Bert, D., Echahed, R., Østvold, B.M.: Abstract Rewriting. In: Cousot, P., Filé, G., Falaschi, M., Rauzy, A. (eds.) WSA 1993. LNCS, vol. 724, pp. 178–192. Springer, Heidelberg (1993)
6. Boudol, G.: Une semantique pour les arbres non deterministes. In: Astesiano, E., Böhm, C. (eds.) CAAP 1981. LNCS, vol. 112, pp. 147–161. Springer, Heidelberg (1981)
7. Boudol, G.: Computational semantics of term rewriting systems. In: Algebraic methods in semantics, pp. 169–236. Cambridge University Press, Cambridge (1986)
8. Braßel, B., Huch, F.: On a tighter integration of functional and logic programming. In: Shao, Z. (ed.) APLAS 2007. LNCS, vol. 4807, pp. 122–138. Springer, Heidelberg (2007)
9. Clavel, M., et al. (eds.): All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007)
10. González-Moreno, J.C., Hortalá-González, T., López-Fraguas, F., Rodríguez-Artalejo, M.: An approach to declarative programming based on a rewriting logic. J. of Logic Programming 40(1), 47–87 (1999)
11. Hanus, M.: Multi-paradigm declarative languages. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 45–75. Springer, Heidelberg (2007)
12. Hanus, M., Lucas, S.: An evaluation semantics for narrowing-based functional logic languages. J. Funct. and Logic Prog. 2001(2) (2001)
13. Hussmann, H.: Non-Determinism in Algebraic Specifications and Algebraic Programs. Birkhäuser Verlag, Basel (1993)
14. López-Fraguas, F., Rodríguez-Hortalá, J., Sánchez-Hernández, J.: Bundles pack tighter than lists. In: Draft Proc. Trends in Funct. Prog, TFP 2007 (2007)
15. Martí-Oliet, N., Meseguer, J.: Rewriting logic: roadmap and bibliography. Theor. Comput. Sci. 285(2), 121–154 (2002)
16. Nyström, S.-O.: There is no fully abstract fixpoint semantics for non-deterministic languages with infinite computations. Inf. Process. Lett. 60(6), 289–293 (1996)
17. Ohlebusch, E.: Advanced topics in term rewriting. Springer, Heidelberg (2002)
18. Plotkin, G.D.: LCF considered as a programming language. Theor. Comput. Sci. 5(3), 225–255 (1977)
19. Reynolds, J.: Theories of Programing Languages. Cambridge University Press, Cambridge (1998)

The Π^0_2-Completeness of Most of the Properties of Rewriting Systems You Care About (and Productivity)

Jakob Grue Simonsen

Department of Computer Science, University of Copenhagen
Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark
simonsen@diku.dk

Abstract. Most of the standard pleasant properties of term rewriting systems are undecidable; to wit: local confluence, confluence, normalization, termination, and completeness.

Mere undecidability is insufficient to rule out a number of possibly useful properties: For instance, if the set of normalizing term rewriting systems were recursively enumerable, there would be a program yielding "yes" in finite time if applied to any normalizing term rewriting system.

The contribution of this paper is to show (the uniform version of) each member of the list of properties above (as well as the property of being a *productive* specification of a stream) complete for the class Π^0_2. Thus, there is neither a program that can enumerate the set of rewriting systems enjoying any one of the properties, nor is there a program enumerating the set of systems that do not.

For normalization and termination we show both the ordinary version and the ground versions (where rules may contain variables, but only ground terms may be rewritten) Π^0_2-complete. For local confluence, confluence and completeness, we show the ground versions Π^0_2-complete.

1 (Uniform) Undecidability in Term Rewriting

It is well-known that almost all of the usual nice-to-have properties of term rewriting systems are undecidable [11,2,18], in particular normalization, termination, local confluence, confluence, and completeness are undecidable, even when rewrite steps are allowed only on ground terms.

Upon inspection, many of the proofs used for undecidability employ reduction from well-known undecidable problems that are semidecidable—technically, the problems are "only" Σ^0_1-complete—the canonical examples being (1) the Halting Problem: the set of integers m encoding pairs $m = \langle M, n\rangle$ such that Turing machine M halts on input n, and (2) the Post Correspondence Problem (PCP) [18, Ch.5].

A standard fact is that Σ^0_1 contains exactly the recursively enumerable sets; hence, if some property is proven undecidable via a reduction from, say, PCP, it could still be possible that the set of objects enjoying the property could be

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 335–349, 2009.
© Springer-Verlag Berlin Heidelberg 2009

recursively enumerable, hence semi-decidable. In rewriting, two simple examples of such a property are (1) for each term rewriting system (TRS) R, the set of terms that have an R-normal form, and (2) the set of TRSs R for which there exists at least one term having a normal form.

However, it would go against all common sense that, say, the set of all (finite) TRSs that are terminating should be semi-decidable: We know that TRSs may simulate Turing machines, and validating whether all terms of some specific TRS are terminating intuitively must consist of checking an infinite number of terms for each TRS, thus requiring an infinite amount of work to validate termination.

The purpose of this paper is to show that several of the usual nice-to-have properties of TRSs are Π_2^0-complete. As neither any semi-decidable set, nor a co-semi-decidable set can be Π_2^0-hard, we thus corroborate the above intution.

We consider so-called *uniform* decision problems: All problems consist of deciding whether a given TRS has some property. This is in contrast to the non-uniform or *local* problems where, given a pair $\langle R, s\rangle_R$ consisting of a TRS R and a term s of R, the problem consists of deciding whether s satisfies some property under R-reduction.

We have endeavoured to show Π_2^0-completeness of the properties when only *ground* terms may be rewritten. This is not to be confused with decision problems for *ground* TRS where, in addition, all *rules* are ground—in that case, many of the properties are decidable [9,12,4]. Furthermore, for the properties of local confluence and confluence, we have succeeded in proving Π_2^0-completeness *only* in the case of ground terms.

Our results depend crucically on the finiteness of all systems and alphabets considered; allowing for systems with, say, an infinite number of rules would propel hardness into the low tiers of the analytical hierarchy instead.

2 Preliminaries

Term rewriting systems (TRSs) (Σ, R) are defined in the usual way [11,18], as is the set of terms. In this paper, Σ and R will always be finite. We usually suppress the alphabet Σ and refer to the TRS merely by the set of rules R.

Definition 1. *A* constructor TRS *is a TRS (Σ, R) where $\Sigma = F \cup C$ with $F \cap C = \emptyset$ (F is called the set of* defined symbols*, C the set of* constructor symbols*), and such that every rewrite rule $l \to r \in R$ satisfies $l = f(t_1, \ldots, t_n)$ with $f \in F$ and $t_1, \ldots, t_n$ are terms over the signature C (that is, contain no defined symbols). When the alphabet Σ is presupposed, we usually refer to the TRS (Σ, R) merely by R. The TRS $(R)_0$ consists of the rules of R, but only ground terms may be rewritten. If s is a term, the set of positions of s and notion of subterm at a position are defined as usual. We denote the subterm at position p of s by $s|_p$, and denote the term obtained by replacing the subterm $s|_p$ in s by term t by $s[t]_p$. The* depth *of a rewrite step is the length of the position of the redex contracted. If $n \geq 0$, we write $s|_{\leq n} = t|_{\leq n}$ if $s|_{\leq n}$ and $t|_{\leq n}$ are identical up to and including all positions of length n.*

Definition 2. *A TRS R is* normalizing *(WN) if every term has a normal form,* terminating *(SN) if all maximal reduction sequences from all terms are finite,* locally confluent *(WCR) if, for every fork $t \leftarrow s \rightarrow t'$ there exists a join $t \rightarrow^* s'^* \leftarrow t'$,* confluent *(CR) if, for every fork $t^* \leftarrow s \rightarrow^* t'$ there exists a join $t \rightarrow^* s'^* \leftarrow t'$, and* complete *(COM) if R is both terminating and confluent.*

We say that R is ground-*WN (resp. -SN,-WCR,-CR,-COM) if R is WN on ground terms, that is, if $(R)_0$ is WN (resp. SN, WCR, CR,COM).*

We presuppose basic knowledge of primitive recursive functions and primitive recursive predicates, see e.g. [13]. The reader without prior knowledge may simply view primitive recursive functions as those functions that are computable using for-loops, but without unconstrained recursion or iteration.

It is easy to see that there are primitive recursive functions en- and decoding each (finite) TRS as an integer, en- and decoding pairs of integers, en- and decoding finite reduction sequences in a given TRS as an integer, and checking whether a given sequence of rewrite steps are allowed by the TRS. For ease of notation, we overload the symbols $\langle\cdot\rangle$ to mean any primitive recursive encoding and $\langle x\rangle_R$ to mean a primitive recursive encoding of x in TRS R, as appropriate by context. Thus, $\langle s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_m\rangle_R$ is the integer encoding the reduction $s_0 \rightarrow s_1 \rightarrow \cdots \rightarrow s_m$ in TRS R, and $\langle R\rangle$ is an integer encoding the TRS R.

2.1 Turing Machines

We refer to standard textbooks [13,16] for a standard treatment of Turing machines. We introduce basic notation:

Definition 3. *A (deterministic, single-tape, binary alphabet)* Turing machine *M is a triple $(Q, \{1, 0, \Box\}, \delta)$ where*

- *Q is a finite set of* states *containing at least two distinct states q_s and q_h.*
- *$\{1, 0, \Box\}$ is the set of* tape symbols *where $\Box$ is interpreted as "blank".*
- *δ is a partial function from $Q \times \{1, 0, \Box\}$ to $Q \times \{1, 0, \Box\} \times \{L, R\}$ and is called the* transition relation. *L represents a move to the left and R a move to the right.*

We assume that for every $q \neq q_h$ and every $s \in \{1, 0, \Box\}$, $\delta(q, s)$ is defined (if $\delta(q, s)$ is undefined, the machine has halted, and we may wlog. add $\delta(q, s) \rightarrow (q_h, \Box, L)$ without affecting the halting behaviour of the machine). Furthermore, we assume wlog. that the machine never returns to its starting state, i.e. that if $\delta(q, s) = (q', s', D)$, then $q' \neq q_s$.

Definition 4. *A* configuration *of a Turing machine M is a triple wqw' where q is a state of M, w and w' are the tape contents to the left and right of the tape head (we assume the head is on the first cell of w'), and we assume that w and w' contain only a finite number of non-blank symbols.*

A start configuration *of M is a configuration wqw' where $q = q_s$, w contains only blanks, and w' consists of a finite number of* consecutive *1s (that is, the Turing machine is in the start state, there is a unary representation of a natural number to the right of the tape head, and nothing else on the tape).*

2.2 The Arithmetical Hierarchy and Π_2^0

Recall that the *arithmetical* (or *Kleene*) hierarchy is a classification of formulae of first-order (Peano) arithmetic; in particular, a formula ϕ is a Π_2^0-formula (or $\forall\exists$-formula) if it is logically equivalent to a formula on the form $\forall n \exists k.P(n,k,x)$ where $P(n,k,x)$ is a primitive recursive predicate.

Definition 5. *The class Π_2^0 comprises of all subsets K of the natural numbers such that there is a Π_2^0-formula that is valid exactly on the elements of K.*

Details on Π_2^0 and other classes of the arithmetical hierarchy may be found in [13]. Note that replacing the demand that $P(n,k,x)$ be primitive recursive by "$P(n,k,x)$ is a decidable predicate" does not change the class Π_2^0. As with complexity classes such as P and NP , we may define the relation $\leq_m$ (many-one reducibility) on Π_2^0:

Definition 6. *The set A* many-one reduces *to set B, written $A \leq_m B$, if there exists a Turing machine M that on input $x \in \mathbb{N}$ outputs $\phi_M(x)$ such that $\phi_M(x) \in B$ iff $x \in A$. We say that B is* Π_2^0-hard *if every $A \in \Pi_2^0$ satisfies $A \leq_m B$), and that B is* Π_2^0-complete *if $B \in \Pi_2^0$ and B is Π_2^0-hard.*

Intuitively, a Π_2^0-hard set is *at least as difficult* to decide as any other set in Π_2^0. Ordinarily, the notion of *Turing reducibility* $\leq_T$ is used in connection with Π_2^0; it is easy to see that $A \leq_m B$ implies $A \leq_T B$, whence our hardness results also holds in the setting of $\leq_T$.

Definition 7. UNIFORM *is the set of (Gödel numbers of) Turing machines that reach the halting state from all configurations.*

TOTALITY *is the set of (Gödel numbers of) Turing machines that reach the halting state from all start configurations.*

Thus, membership in UNIFORM requires that M halts on any configuration (even ones unreachable from the start state), while membership in TOTALITY requires that M halts when run on any natural number.

Proposition 1. *Both* UNIFORM *and* TOTALITY *are Π_2^0-complete.*

Proof. Standard. See e.g. [13, Ch. 13–14] for TOTALITY, and [8] for UNIFORM. □

By standard results for the arithmetical hierarchy, no Π_2^0-hard set is semi-decidable, nor is it co-semi-decidable.

3 Encoding Turing Machines

The paper uses three encodings of Turing machines M. Of these encodings, the first—$\Delta(M)$—is the standard encoding of M as an orthogonal TRS from [18, Ch. 5], itself a clever variation of the idea in [9]. The second encoding—$\Delta_{\mathrm{g}}(M)$—adds a number function symbols and rules to $\Delta(M)$ to handle the ground-WCR

and ground-CR properties and is, by construction, non-orthogonal. The third encoding—$\Delta_S(M)$—extends $\Delta(M)$ by letting certain function symbols take an extra argument (which is unchanged by the rewrite rules) and adding two new simple rules, obtaining an orthogonal constructor TRS.

We now give the encoding $\Delta(M)$ of [18, Ch. 5]. A Turing machine M is encoded as a TRS $\Delta(M)$ as follows: Each state $q \in Q$ of the machine M is coded as a binary function symbol q. The alphabet of $\Delta(M)$ furthermore contains unary function symbols 1, 0 and $\Box$ corresponding to the elements of the tape alphabet of M, and a nullary function symbol $\triangleright$ representing an infinite stretch of blanks ($\triangleright$ may thus be thought of as an "endmarker" demarcating the finite part of the tape containing non-blank symbols). A word $w \in \{0, 1, \Box\}^*$ is translated into a term $\phi(w)$ by setting $\phi(\epsilon) \triangleq \triangleright$ (where ϵ is the empty word), and $\phi(bw) \triangleq b(\phi(w))$ for $b \in \{1, 0, \Box\}$ and $w \in \{1, 0, \Box\}^*$. When $b_1, \ldots, b_n$ are unary function symbols and t is a term, we write $b_1 b_2 \cdots b_n t$ instead of $b_1(b_2(\cdots(b_n(t))))$. A configuration $w_1 q w_2$ of M is translated to the $\Delta(M)$-term $q(\phi(\text{reverse}(w_1)), \phi(w_2))$ where $\text{reverse}(w_1)$ is w_1 reversed. For ease of notation, we suppress ϕ and reverse and write $q(w_1, w_2)$.

The rules of the encoding is given in Figure 1.

Rewrite rules induced by transition rules of the Turing machine M ($\Delta_N(M)$):

(L/R)-move	rewrite rules (for each $q \in Q$, $a \in \{0, 1, \Box\}$)
$\delta(q, b) = (q', b', R)$	$q(x, by) \to q'(b'x, y)$
$\delta(q, b) = (r', b', L)$	$q(ax, by) \to q'(x, ab'y)$

Extra rules ($\Delta_E(M)$):

(L/R)-move	extra rewrite rules (for each $q \in Q$, $a \in \{0, 1, \Box\}$)
$\delta(q, \Box) = (q', b', R)$	$q(x, \triangleright) \to q'(b'x, \triangleright)$
$\delta(q, b) = (q', b', L)$	$q(\triangleright, by) \to q'(\triangleright, \Box b'y)$
$\delta(q, \Box) = (q', b', L)$	$q(ax, \triangleright) \to q'(x, ab'\triangleright)$ $q(\triangleright, \triangleright) \to q'(\triangleright, \Box b'\triangleright)$

$$\Delta(M) = \Delta_N(M) \cup \Delta_E(M)$$

Fig. 1. Encoding $\Delta(M)$ of a Turing machine M

It is straightforward to show that for all Turing-machine configurations α, β of M, we have (1) if M moves from α M to β in one move and term s represents α, then there is a term t with $s \to t$ and such that t represents β, respectively (2) if term s represents α and $s \to t$, then there is a configuration β such that M moves from α to β in one step and t represents β [18, Exercise 5.3.3].

For later use, we note the following fortuitous property of $\Delta(M)$:

Proposition 2. *Call any ground term of* $\Delta(M)$ restricted *if it is on the form* $q(s, s')$ *where* s *and* t *contain no occurrence of any* $q \in Q$. *Then any restricted*

ground term represents a configuration of M, and if $\Delta(M)$ admits an infinite reduction starting from some term, there is a restricted ground term t having an infinite reduction

Proof. Straightforward. See e.g. [18, Exercise 5.3.3] or [11, Exercise 2.2.12]. □

Corollary 1. *$\Delta(M)$ is both SN and ground-SN iff M halts on all configurations.*

Proof. If there is a configuration wqw' on which M does not halt, then there is an infinite $\Delta(M)$-reduction starting from the ground term $q(w, w')$, showing that $\Delta(M)$ is neither SN, nor ground-SN. If M halts on all configurations, suppose for the purpose of contradiction that $\Delta(M)$ were not SN. By Proposition 2 there is a restricted ground term t having an infinite reduction. But by the same proposition, any restricted ground term represents a configuration wqw' of M, entailing that M does not halt on wqw', a contradiction. □

Corollary 2. *$\Delta(M)$ is both WN and ground-WN iff M halts on all configurations.*

Proof. As the previous corollary. □

3.1 Adding Rules for Ground-WCR and CR: the Encoding $\Delta_{\mathrm{g}}(M)$

For the purpose of proving ground-WCR and ground-CR Π^0_2-hard, we shall need to extend the encoding $\Delta(M)$. We extend the alphabet of $\Delta(M)$ with two new function symbols: the nullary symbol FAIL and the binary symbol A, and we extend the rules of $\Delta(M)$ with a number of new rules shown in Figure 2. The rules are specifically designed for (1) making it possible to rewrite ground terms that do *not* represent a configuration of M to the constant FAIL, and for (2) creating a fork $\text{FAIL} \leftarrow A(w, w') \rightarrow q(w, w)$ that has no corresponding join unless M reaches the halting state from configuration wqw'.

New rewrite rules ($\Delta_{\mathrm{c}}(M)$):

$\rightarrow_{\mathrm{c}}$-rules (for each $q \in Q$)

$0q(y,z) \rightarrow_{\mathrm{c}} \text{FAIL}$	$q(q'(x,y),z) \rightarrow_{\mathrm{c}} \text{FAIL}$
$1q(y,z) \rightarrow_{\mathrm{c}} \text{FAIL}$	$q(x,q'(y,z)) \rightarrow_{\mathrm{c}} \text{FAIL}$
$\Box q(y,z) \rightarrow_{\mathrm{c}} \text{FAIL}$	$q(\text{FAIL},y) \rightarrow_{\mathrm{c}} \text{FAIL}$
$0\text{FAIL} \rightarrow_{\mathrm{c}} \text{FAIL}$	$q(x,\text{FAIL}) \rightarrow_{\mathrm{c}} \text{FAIL}$
$1\text{FAIL} \rightarrow_{\mathrm{c}} \text{FAIL}$	$A(x,y) \rightarrow_{\mathrm{c}} \text{FAIL}$
$\Box\text{FAIL} \rightarrow_{\mathrm{c}} \text{FAIL}$	$A(x,y) \rightarrow_{\mathrm{c}} q(x,y)$
$q_h(x,y) \rightarrow_{\mathrm{c}} \text{FAIL}$	

$$\Delta_{\mathrm{g}}(M) = \Delta(M) \cup \Delta_{\mathrm{c}}(M)$$

Fig. 2. Encoding $\Delta_{\mathrm{g}}(M)$ for handling (local) confluence for ground terms

Remark 1. Observe that if $q(w, w')$ is a term representing a configuration wqw' of M and $q \neq q_h$, then $q(w, w')$ contains no $\Delta_{\mathrm{C}}(M)$-redex (in particular, $q(w, w')$ does not contain any occurrence of FAIL). Furthermore, observe that if $q(s, t)$ is a restricted ground term containing no occurrence of FAIL or A, and if $q(s, t) \to^* q'(s', t')$, then $q'(s', t')$ contains no occurrence of FAIL or A: this is easily seen by the fact that A does not occur on any right-hand side of any rule and the fact that FAIL can only be introduced in one step by a $\Delta_{\mathrm{C}}(M)$-redex; as $q(s, t)$ is restricted ground, the only $\Delta_{\mathrm{C}}(M)$-redex possible in $q(s, t) \to^* q'(s', t')$ is in the final term, and only if $q' = q_h$.

Finally, note that if $q(s, t)$ is a restricted ground term containing no occurrence of FAIL or A, $\Delta_{\mathrm{g}}(M)$ enjoys the same simulation properties as $\Delta(M)$. □

Proposition 3. *If $s \to t$ by an application of a $\Delta_{\mathrm{C}}(M)$-redex at depth $d \geq 1$ in s, then t reduces to* FAIL *in at most d steps.*

Proof. Straightforward induction on d. □

Proposition 4. *$\Delta_g(M)$ is ground-SN iff M halts on all configurations.*

Proof. Straightforward analysis appealing to the fact that $\Delta(M)$ is SN iff M halts on all configurations and the fact that $\Delta_{\mathrm{C}}(M)$ can only create an $\Delta(M)$-redex by the rules in $\{A(x, y) \to_{\mathrm{C}} q(x, y) : q \in Q\}$. □

Proposition 5. *The only ground normal form of $\Delta_g(M)$ is* FAIL.

Proof. Straightforward case analysis using the fact that any ground term that does not represent a configuration of M will either be FAIL or contain a $\Delta_{\mathrm{C}}(M)$-redex. □

Proposition 6. *$\Delta_g(M)$ is ground-WCR iff M halts on all configurations.*

Proof. We proceed as follows.

- "⇒": By contraposition. Suppose there is a configuration wqw' on which M does not halt. Then there is a restricted ground term $q(w, w')$ that represents wqw' (hence contains no occurrence of FAIL or A), and there is an infinite reduction $A(w, w') \to_{\mathrm{C}} q(w, w') \to q'(s, s') \to q''(s'', s''') \to \cdots$. By assumption, no M-step ever reaches the halting state, whence no reduct of $q(w, w')$ can contain an occurrence of q_h . Furthermore, by Remark 1, no reduct of $q(w, w')$ contains any occurrence of FAIL or A, whence no term in $q(w, w') \to q'(s, s') \to q''(s'', s''') \to \cdots$ can reduce to FAIL. But $A(w, w') \to_{\mathrm{C}}$ FAIL and $A(w, w') \to_{\mathrm{C}} q(w, w')$, showing that $\Delta_{\mathrm{g}}(M)$ is not ground-WCR (and not WCR).
- "⇐": Suppose M halts on all configurations, hence that $\Delta_{\mathrm{g}}(M)$ is ground-SN by Proposition 4 . Thus, for any fork $t \leftarrow s \to t'$, both t and t' have normal forms t_f and t'_f. As s is ground, so are t and t', and Proposition 5 yields $t_f = t'_f =$ FAIL, whence $t \to^* \mathrm{FAIL}^* \leftarrow t'$, as desired.

□

Corollary 3. *$\Delta_g(M)$ is ground-CR iff M halts on all configurations.*

Proof. If $\Delta_{\mathrm{g}}(M)$ is ground-CR, it is in particular ground-WCR, hence by Proposition 6 M terminates on all inputs. Conversely, if M terminates on all inputs, Propositions 6 and 4 yield that $\Delta_{\mathrm{g}}(M)$ is WCR and SN, respectively. Confluence of $\Delta_{\mathrm{g}}(M)$ follows by Newman's Lemma. □

Corollary 4. *$\Delta_g(M)$ is ground-COM iff M halts on all configurations.*

Proof. By Proposition 4 and Corollary 3. □

We postpone the final encoding of M, $\Delta_{\mathrm{S}}(M)$, to Section 5.

4 Π_2^0-Completeness of the Standard Properties

In the below, we consider ternary primitive recursive predicates $P(n,k,l)$ in Π_2^0-formulae $\forall n \exists k.P(n,k,l)$ where l is thus the only free variable. As all the decision problems we consider take a TRS as "input", the variable l will always be an integer encoding a TRS R. In all predicates $P(n,k,l)$, P decodees integers n and k to terms of R or finite R-reductions, and perform simple checks. For ease of notation, instead of referring explicitly to l, we will use the obvious shorthands $\langle\cdot\rangle_R$ and $\langle\cdot,\cdot\rangle_R$ in the predicates.

4.1 (Ground-)Local Confluence

The usual "easy" proofs of undecidability of (local) confluence employ reductions from the (local) word problem for semi-groups (which is in Σ_1^0 and hence is not Π_2^0-complete), see e.g. [18]. More detailed analyses reveal that neither the uniform problem of whether a TRS is ground-WCR, nor the local problem of whether a term in a TRS is ground-WCR or ground-CR is semi-decidable [10]. However, even with these results, (ground-)WCR or (ground-)CR could still be co-semi-decidable. We will rule out this possibility by proving both problems Π_2^0-complete using $\Delta_{\mathrm{g}}(M)$. We do not know if our results can be extended to encompass WCR and CR: the encoding $\Delta_{\mathrm{g}}(M)$ is not sufficient to do the trick as non-ground terms that do not represent configurations of M may be normal forms of $\Delta_{\mathrm{g}}(M)$, rendering our proof methods unapplicable.

Proposition 7. *WCR and ground-WCR are both Π_2^0-properties of TRSs.*

Proof. Using the primitive recursive en- and decoding of finite reduction sequences as integers, we may write the property of the TRS (Σ, R) being locally confluent as:

$$\forall n \exists k. \begin{pmatrix} n = \langle s \to s', s \to s'' \rangle_R & \\ k = \langle s' \to s'_1 \to \cdots \to s'_l, s'' \to s''_1 \to \cdots \to s''_m \rangle_R & \Rightarrow \end{pmatrix}$$

□

Theorem 1. *Deciding whether a TRS is ground-WCR is Π_2^0-complete.*

Proof. There is clearly a recursive procedure transforming a description of a Turing machine M into the finite TRS $\Delta_{\mathrm{g}}(M)$. By Proposition 6, this is a many-one reduction from UNIFORM to the set of TRSs that are ground-WCR, showing that deciding whether a given TRS is $(-)_0$wcr is Π^0_2-hard. Completeness now follows from Proposition 7. □

4.2 (Ground-)Confluence

In analogy to WCR, we have:

Proposition 8. *CR and ground-CR are both Π^0_2-properties.*

Proof. As with local confluence we may write the property of TRS (Σ, R) being confluent as:

$$\forall n \exists k. \begin{pmatrix} n = \langle s \to s_1 \to \cdots \to s_l, s \to s'_1 \to \cdots \to s'_m \rangle_R \\ k = \langle s_l \to s_{l+1} \to \cdots \to s_{l+r}, s'_m \to s'_{m+1} \to \cdots \to s'_{m+p} \rangle_R \end{pmatrix} \Rightarrow$$

□

Theorem 2. *Deciding whether a TRS is ground-CR is Π^0_2-complete.*

Proof. By Corollary 3 and Proposition 8, reasoning as in the proof of Theorem 1. □

4.3 Normalization

Both WN and SN have previously been shown Π^0_2-hard, even though the results were never explicitly stated—in fact, the encoding $\Delta(M)$ and its use in [18] to prove SN undecidable is in fact a reduction from UNIFORM to SN. We explicitly state the results below.

Proposition 9. *WN and ground-WN are both Π^0_2-properties of TRSs.*

Proof. The predicate "term s is a normal form of (finite) TRS R" is primitive recursive; we write $\mathrm{nf}(s)_R$ for the predicate. Using primitive recursive en- and decoding of terms and finite reductions sequences, we may write the property of TRS (Σ, R) being normalizing as:

$$\forall n \exists k. (n = \langle s \rangle_R \Rightarrow (k = \langle s \to s_1 \to \cdots \to s_m \rangle_R \wedge \mathrm{nf}(s_m)_R))$$

□

Theorem 3. *Deciding whether a TRS is WN (resp. ground-WN) is Π^0_2-complete.*

Proof. By Corollary 2 and Proposition 9, reasoning as in the proof of Theorem 1. □

4.4 Termination

Unlike the previous properties, a technical snag of proving Π_2^0-completeness of termination is to show that the problem is *included* in Π_2^0: Termination means that *for all* terms s, there are *no* infinite reduction sequences starting from s. Neither this phrasing, nor the phrasing using finiteness of maximal reductions, is amenable to putting in $\forall\exists$-form. We work around this problem by appealing to König's Lemma.

Definition 8. *Let s be a term. The* derivation tree *of s is the (possibly infinite) rooted tree $\mathcal{G}(s)$ such that (i) level 0 in the tree contains the term s, and (ii) for $n \geq 0$, the nodes of level $n+1$ are such that, for each node s at level n, there is a node t at level $n+1$ and an edge from s to t iff $s \to t$.*

Nodes at level $n+1$ are not shared, that is, a term t may occur as several different nodes if it is a reduct of several different terms at level n.

Proposition 10. *The term s is terminating iff $\mathcal{G}(s)$ is finite.*

Proof. $\mathcal{G}(s)$ is finitely branching as each term has a finite number of reducts. König's Lemma now yields that $\mathcal{G}(s)$ is infinite iff there exists an infinite path in $\mathcal{G}(s)$. By construction of $\mathcal{G}(s)$, s is terminating iff there exists an infinite path in $\mathcal{G}(s)$, concluding the proof. □

Observe that any *finite* tree labeled with terms over alphabet Σ can be en- and decoded as an integer by a primitive recursive function. If G is such a finite tree, we shall denote by $\langle G \rangle_R$ its encoding. Checking that $\langle G \rangle_R$ encodes the derivation tree $\mathcal{G}(s)$ of a term s is again primitive recursive as the first n levels of $\mathcal{G}(s)$ may be generated by a primitive recursive procedure: Exhaustively try all rules in the term rewriting system R on all positions of s to obtain level 1 in the graph and proceed iteratively until n levels have been processed. Given $\langle G \rangle_R$, obtain its maximum depth n by a straightforward search, then generate $\mathcal{G}(s)$ to depth n and call the result $\mathcal{G}(s)|_n$. We then have $G = \mathcal{G}(s)$ iff all leaves in $\mathcal{G}(G)|_n$ are normal forms and $G = \mathcal{G}(G)|_n$. Let the primitive recursive predicate that $G = \mathcal{G}(s)$ be $P(\langle s \rangle_R, \langle G \rangle_R)$—note that $\langle R \rangle$ is a suppressed third variable of P.

Proposition 11. *SN and ground-SN are both Π_2^0-properties.*

Proof. Using $\langle G \rangle_R$ we may write the property of TRS (Σ, R) being terminating as:

$$\forall n \exists k.\, (n = \langle s \rangle_R \wedge k = \langle G \rangle_R \wedge P(n,k))$$

For ground-SN, all that needs to be changed is to change the encoding $\langle \cdot \rangle_R$ from arbitrary terms to ground ones. □

Theorem 4. *Deciding whether a TRS is SN (resp. ground-SN) is Π_2^0-complete.*

Proof. By Corollary 1 and Proposition 11, reasoning as in the proof of Theorem 1. □

4.5 Completeness

Theorem 5. *Deciding whether a TRS is ground-COM is Π^0_2-complete.*

Proof. If A and B are both sets in Π^0_2, then so is $A \cap B$. Thus, by Propositions 8 and 11, ground-COM is a Π^0_2-property. Hardness of ground-COM follows from the fact that $\Delta_{\mathrm{C}}(M)$ is ground-SN and ground-CR iff M halts on all configurations by Proposition 4 and Corollary 3. □

5 Π^0_2-Completeness of Productivity (of Stream Specifications)

A *stream specification* is intuitively a program that lazily outputs a list. If such a program, when run for a sufficient amount of time, always outputs "the next" element of the list, it is called *productive*. A substantial amount of work aims at establishing sufficient conditions for establishing the productivity of stream specifications [5,15,17,3,7,6]. It is well-known folklore—and almost self-evident—that the problem of establishing whether a stream specification is productive is undecidable. Some of the recent literature on stream definitions employ a complicated but clean notion of stream specification stratified into data, function and stream layer and using many-sorted rewriting [7,6]. Here, we consider the more generic approach where a specification is simply a constructor TRS with a designated symbol for the stream.

Definition 9. *A* stream specification *is a pair $((F \cup C, R), S)$ consisting of a constructor TRS $(F \cup C, R)$ and a designated nullary defined symbol $S \in F$. The stream specification is said to be* productive *if S has an infinite constructor normal form, that is, if there exists an infinite reduction $S \to^* t_1 \to^* t_2 \to^* \cdots$ such that increasingly larger prefixes of the terms t_n consist solely of constructor symbols. The stream specification is said to be* orthogonal *if $(F \cup C, R)$ is orthogonal.*

We note that several "streams" may be defined by the same definition; however we are only interested in *one* of these, namely the one designated by S. Most literature on productivity focuses specifically on the case where the data modelled are truly streams of bits, that is, $C = C_{\mathrm{S}} \triangleq \{:, \mathbf{0}, \mathbf{1}\}$. We note that the *hardness* proof below specifically uses only C_{S}—in fact, uses only $\{:, \mathbf{1}\}$—thus showing that even a very restricted constructor alphabet corresponding to a classical "lazy list of bits" implies Π^0_2-hardness. A full account of infinite (constructor) terms is beyond the scope of this paper, and we refer to [1,18]. For our purposes, the reader need only consider the set of infinite (constructor) terms over C_{S}, for example $\mathbf{1}{:}\mathbf{1}{:}\mathbf{1}{:}\cdots$. We invariably write the binary symbol : as infix, and if s is a term, we use $\mathbf{1}{:}\mathbf{1}{:}\cdots{:}\mathbf{1}{:}s$ as shorthand for $\mathbf{1}{:}(\mathbf{1}{:}(\cdots\mathbf{1}(:s)))$.

We note the following fact about orthogonal constructor TRSs:

Proposition 12. *If $(F \cup C, R)$ is orthogonal, then $((F \cup C, R), S)$ is productive iff for all $n \geq 0$ there is a finite rewrite sequence $S \to^* t_n$ where t is a term on the form $C[s_1, \ldots, s_m]$ with $C[x_1, \ldots, x_m]$ a constructor term of depth n.*

Proof. Straightforward analysis using the fact that $(F \cup C, R)$ is orthogonal, hence is confluent, and that no constructor symbol is at the root of the left-hand side of a rewrite rule $l \to r \in R$. □

Example 1. The assumption of orthogonality cannot be omitted from Proposition 12. Consider for instance the below (non-orthogonal) constructor TRS (constructor symbols are boldface):

$$\left\{ \begin{array}{c} S \to a(\mathbf{b}) \\ a(\mathbf{b}) \to a(a(\mathbf{b})) \\ a(\mathbf{b}) \to \mathbf{c}(\mathbf{b}) \\ a(\mathbf{c}(x)) \to \mathbf{c}(\mathbf{c}(x)) \end{array} \right\}$$

For every $n \geq 1$ we have $S \to^* \underbrace{\mathbf{c}(\mathbf{c}(\cdots(\mathbf{c}}_{n}(\mathbf{b}))))$ which is a constructor normal form. However, S has no infinite constructor normal form. □

Proposition 13. *Productivity of orthogonal stream specifications is a Π_2^0-property.*

Proof. Observe that there is a primitive recursive function that, when input n and an integer j encoding some term s checks whether s contains only constructor symbols up to depth n (simply use a top-down iteration on s up to depth n). Let $P(n, j, l)$ be the primitive recursive predicate checking whether the above is true for orthogonal stream specification $l = \langle((F \cup C, R), S)\rangle_R$.

Thus,

$$\forall n \exists k. \left(k = \langle S \to s_1 \to \cdots \to s_m \rangle_R \wedge P(n, \langle s_m \rangle, l) \right)$$

is a Π_2^0-formula which, by Proposition 12 is true iff $((F \cup C, R), S)$ is productive. □

We change the encoding of Turing machines M slightly to accomodate productivity; the new encoding is shown in Figure 3.Each state q is now encoded by a *ternary* function symbol such that in $q(s, t, t')$, s and t are the contents of the left and right part of the tapes as usual, and t' represents the "current" input being processed by M. We *change* the rules of Figure 1 by letting the third argument t' of each $q(s, t, t')$ be identical in left- and right-hand sides (that is, the argument is unchanged by applying any of the rules). We add two new rules: the rule $q_h(x, y, z) \to \mathbf{1}{:}q_s(\triangleright, 1z, 1z)$ that "produces" $\mathbf{1}$, and the rule $S \to q_s(\triangleright, 1\triangleright, 1\triangleright)$ that initiates the computation. Thus, the translation $\Delta_{\mathrm{S}}(M)$ of M produces a stream specification $((F \cup C, R), S)$ where S is a some fresh symbol, $F = \{S\} \cup \{0, 1, \square, \triangleright\} \cup Q$ where Q contains a ternary symbol for each state of M, where $C = \{\mathbf{0}, \mathbf{1}, :\}$ (where the nullary constructor symbols $\mathbf{0}$ and $\mathbf{1}$ are not to be confused with the unary defined symbols 0 and 1), and R is the set of rules obtained in Figure 3.

Remark 2. Call a ground term of $\Delta_{\mathrm{S}}(M)$ restricted if it is on the form $q(w, w', t)$ with $q \neq q_h$ and where w, w, t terms over F such that no q occurs in w, w', or t.

Rewrite rules induced by transition rules of the Turing machine M ($\mathcal{A}'_N(M)$):

(L/R)-move	rewrite rule (for each $q \in Q$, $a \in \{0, 1\Box\}$)
$\delta(q,b) = (q', b', R)$	$q(x, sy, z) \to q'(b'x, y, z)$
$\delta(q,b) = (q', b', L)$	$q(ax, by, z) \to q'(x, ab'y, z)$

Extra rules ($\mathcal{A}'_E(M)$)):

(L/R)-move	extra rewrite rules (for each $q \in Q$, $a \in \{0, 1\Box\}$))
$\delta(q,\Box) = (q', b', R)$	$q(x, \rhd, z) \to q'(b'x, \rhd, z)$
$\delta(q,b) = (q', b', L)$	$q(\rhd, by, z) \to q'(\rhd, \Box b'y, z)$
$\delta(q,\Box) = (q', b', L)$	$q(ax, \rhd, z) \to q'(x, ab'\rhd, z)$ $q(\rhd, \rhd, z) \to q'(\rhd, \Box b'\rhd, z)$

New rules for transitions from the halting state ($\mathcal{A}'_{\mathrm{p}}(M)$):

new rules for stream productivity)
$q_h(x, y, z) \to \mathbf{1}{:}q_s(\rhd, 1z, 1z)$
$S \to q_s(\rhd, 1\rhd, 1\rhd)$

$$\mathcal{A}_{\mathrm{S}}(M) = \mathcal{A}'_N(M) \cup \mathcal{A}'_E(M) \cup \mathcal{A}'_{\mathrm{p}}(M)$$

Fig. 3. Encoding $\mathcal{A}_{\mathrm{S}}(M)$ of Turing machines for stream productivity

As $\mathcal{A}_{\mathrm{S}}(M)$ is contructed exactly as $\mathcal{A}(M)$, bar $\mathcal{A}'_{\mathrm{p}}(M)$ and the increased arity of the state symbols q, we immediately see that if α and β are configurations of M such that M moves from α to β and $q(w, w', t)$ represents α, then there is a term s' with $s \to s'$, and if $q(w, w, t)$ represents configuration α and $s \to s'$, then s' represents a configuration β such that M moves from α to β. □

Proposition 14. *$(S, \mathcal{A}_S(M))$ is productive iff M halts on all inputs.*

Proof. Reason as follows:

– "⇒": The only possible reduction from S is $S \to q_s(\rhd, 1\rhd, 1\rhd)$. By inspection of the rules, the only way of replacing a function symbol by a constructor is by application of the rule $q_h(x, y, z) \to \mathbf{1}{:}q_s(\rhd, 1z, 1z)$. Thus, the only possibly infinite constructor normal form of S is $\mathbf{1}{:}\mathbf{1}{:}\mathbf{1}{:}\cdots$. We show by induction on $n \geq 1$ that:

$$\begin{aligned} S &\to q_s(\rhd, 1\rhd, 1\rhd) \\ &\to^* q_h(s_1, t_1, 1\rhd) \\ &\to \mathbf{1} : q_s(\rhd, 11\rhd, 11\rhd) \\ &\to^* \underbrace{\mathbf{1}{:}\cdots\mathbf{1}}_{n-1}{:}q_h(s_2, t_2, 1^n\rhd) \\ &\to^* \underbrace{\mathbf{1}{:}\cdots{:}\mathbf{1}}_{n}{:}q_s(\rhd, 1^{n+1}\rhd, 1^{n+1}\rhd) \end{aligned}$$

The base case $n = 1$ follows immediately by inspection of the rules of $\mathcal{A}_{\mathrm{S}}(M)$: The only way to obtain a term on the form $\mathbf{1}{:}s$ is by an application of the rule $q_h(x, y, z) \to \mathbf{1}{:}q_s(\rhd, 1z, 1z)$. As $q_s(\rhd, 1\rhd, 1\rhd)$ is a restricted ground term,

every rewrite step of $\Delta_{\mathrm{S}}(M)$ that is *not* of rule $q_h(x,y,z) \to \mathbf{1}{:}q_s(\triangleright, 1z, 1z)$ or $S \to q_s(\triangleright, 1\triangleright, 1\triangleright)$ simulates a step of M. Hence, there is a reduction $S \to q_s(\triangleright, 1\triangleright, 1\triangleright) \to^* q_h(s_1, t_1, 1\triangleright) \to \mathbf{1} : q_s(\triangleright, 11\triangleright, 11\triangleright)$, as desired.

For the case $n > 1$, the induction hypothesis yields that:

$$S \to^* \underbrace{\mathbf{1}{:}\cdots{:}\mathbf{1}}_{n-1}{:}q_s(\triangleright, 1^n\triangleright, 1^n\triangleright)$$

By Proposition 12, productivity of $\Delta_{\mathrm{S}}(M)$ yields that $S \to^* \underbrace{\mathbf{1}{:}\cdots{:}\mathbf{1}}_{n}{:}t$ for some term t, and orthogonality (hence confluence) of $\Delta_{\mathrm{S}}(M)$ yields that $\mathbf{1}{:}\cdots{:}\mathbf{1}{:}q_s(\triangleright, 1^n, 1^n)$ and $\mathbf{1}{:}\cdots{:}\mathbf{1}{:}t$ have a common reduct which, as $\Delta_{\mathrm{S}}(M)$ is a constructor TRS must be on the form $\underbrace{\mathbf{1}{:}\cdots{:}\mathbf{1}}_{n}{:}t'$ for some term t'. Thus, we have $q_s(\triangleright, 1^n, 1^n) \to \mathbf{1}{:}t'$, and reasoning as in the base case, we obtain $q_s(\triangleright, 1^n, 1^n) \to^* q_h(s', s'', 1^{n+1}\triangleright) \to \mathbf{1}{:}q_s(\triangleright, 1^{n+1}\triangleright, 1^{n+1}\triangleright)$.

Thus, for all n, we have $q_s(\triangleright, 1^n\triangleright, 1^n\triangleright) \to^* \mathbf{1}{:}q_h(s'_n, s''_n, 1^{n+1}\triangleright)$ by a reduction that uses no rules from $\Delta'_{\mathrm{p}}(M)$. As each step in the reduction simulates a step of M, we conclude that M halts on all inputs.

– "$\Leftarrow$": If M halts on all inputs, we have in particular for all $n \geq 1$ that $q_s(\triangleright, 1^n\triangleright, 1^n\triangleright) \to^* q_h(s_n, t_n, 1^n\triangleright)$ for terms s_n, t_n, and hence that:

$$\begin{aligned}\underbrace{\mathbf{1}{:}\cdots{:}\mathbf{1}}_{n-1}{:}q_s(\triangleright, 1^n\triangleright, 1^n\triangleright) &\to^* \underbrace{\mathbf{1}{:}\cdots{:}\mathbf{1}}_{n-1}{:}q_h(s_n, t_n, 1^n\triangleright)\\ &\to \underbrace{\mathbf{1}{:}\cdots{:}\mathbf{1}}_{n}{:}q_s(\triangleright, 1^{n+1}\triangleright, 1^{n+1}\triangleright)\end{aligned}$$

for all $n \geq 1$. Hence, S reduces to the infinite normal form $\mathbf{1}{:}\mathbf{1}{:}\cdots$, as desired. □

Proposition 15. *Deciding whether an orthogonal stream specification is productive is Π^0_2-hard.*

Proof. By reduction from TOTALITY. Obviously, there is an effective procedure transform a description of any Turing machine M into the stream specification $(S, \Delta_{\mathrm{S}}(M))$. The result now follows by Proposition 14. □

Theorem 6. *Deciding whether an orthogonal stream specification is productive is Π^0_2-complete.*

Proof. By Propositions 13 and 15, reasoning as in the proof of Theorem 1. □

In the proof of membership of Π^0_2 (Proposition 13, we have crucially employed the assumption of orthogonality. We do not know if productivity of arbitrary stream specifications is in Π^0_2. Roşu has proved that deciding equality of streams is Π^0_2-complete (in an equational setting, but his result carries over to orthogonal stream specifications) [14]; it is also unknown whether that result holds for non-orthogonal stream specifications.

References

1. Arnold, A., Nivat, M.: The metric space of infinite trees. algebraic and topological properties. Fundamenta Informaticae 3(4), 445–476 (1980)
2. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
3. Buchholz, W.: A term calculus for (co-)recursive definitions on streamlike data structures. Annals of Pure and Applied Logic 136(1–2), 75–90 (2005)
4. Dauchet, M., Heuillard, T., Lescanne, P., Tison, S.: Decidability of the confluence of finite ground term rewrite systems and of other related term rewrite systems. Information and Computation 88(2), 187–201 (1990)
5. Dijkstra, E.W.: On the productivity of recursive definitions. EWD 749 (1980)
6. Endrullis, J., Grabmayer, C., Hendriks, D.: Data-oblivious stream productivity. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS, vol. 5330, pp. 79–96. Springer, Heidelberg (2008)
7. Endrullis, J., Grabmayer, C., Hendriks, D., Isihara, A., Klop, J.W.: Productivity of stream definitions. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 274–287. Springer, Heidelberg (2007)
8. Herman, G.T.: Strong computability and variants of the uniform halting problem. Zeitschrift für mathematische Logik und Grundlagen der Mathematik 17(1), 115–131 (1971)
9. Huet, G., Lankford, D.S.: On the uniform halting problem for term rewriting systems. Rapport Laboria 283, IRIA (1978)
10. Klay, F.: Undecidable properties of syntactic theories. In: Book, R.V. (ed.) RTA 1991. LNCS, vol. 488, pp. 136–149. Springer, Heidelberg (1991)
11. Klop, J.W.: Term rewriting systems. In: Abramsky, S., Gabbay, D., Maibaum, T. (eds.) Handbook of Logic in Computer Science, vol. 2, pp. 1–116. Oxford University Press, Oxford (1992)
12. Oyamaguchi, M.: The Church-Rosser property for ground term rewriting systems is decidable. Theoretical Computer Science 49(1), 43–79 (1987)
13. Rogers Jr., H.: Theory of Recursive Functions and Effective Computability. MIT Press, Cambridge (1987) (paperback edition)
14. Roşu, G.: Equality of streams is a Π_2^0-complete problem. In: Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming (ICFP 2006), pp. 184–191. ACM Press, New York (2006)
15. Sijtsma, B.A.: On the productivity of recursive list definitions. ACM Transactions on Programming Languages and Systems 11(4), 633–649 (1989)
16. Sipser, M.: Introduction to the Theory of Computation. Thomson Course Technology, 2nd edn. (2006)
17. Telford, A., Turner, D.: Ensuring streams flow. In: Johnson, M. (ed.) AMAST 1997. LNCS, vol. 1349, pp. 509–523. Springer, Heidelberg (1997)
18. Terese: Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press, Cambridge (2003)

Unification in the Description Logic $\mathcal{EL}$

Franz Baader and Barbara Morawska

Theoretical Computer Science, TU Dresden, Germany
{baader,morawska}@tcs.inf.tu-dresden.de

Abstract. The Description Logic $\mathcal{EL}$ has recently drawn considerable attention since, on the one hand, important inference problems such as the subsumption problem are polynomial. On the other hand, $\mathcal{EL}$ is used to define large biomedical ontologies. Unification in Description Logics has been proposed as a novel inference service that can, for example, be used to detect redundancies in ontologies. The main result of this paper is that unification in $\mathcal{EL}$ is decidable. More precisely, $\mathcal{EL}$-unification is NP-complete, and thus has the same complexity as $\mathcal{EL}$-matching. We also show that, w.r.t. the unification type, $\mathcal{EL}$ is less well-behaved: it is of type zero, which in particular implies that there are unification problems that have no finite complete set of unifiers.

1 Introduction

Description logics (DLs) [5] are a family of logic-based knowledge representation formalisms, which can be used to represent the conceptual knowledge of an application domain in a structured and formally well-understood way. They are employed in various application domains, such as natural language processing, configuration, databases, and biomedical ontologies, but their most notable success so far is the adoption of the DL-based language OWL [15] as standard ontology language for the semantic web.

In DLs, concepts are formally described by *concept terms*, i.e., expressions that are built from concept names (unary predicates) and role names (binary predicates) using concept constructors. The expressivity of a particular DL is determined by which concept constructors are available in it. From a semantic point of view, concept names and concept terms represent sets of individuals, whereas roles represent binary relations between individuals. For example, using the concept name Woman, and the role name child, the concept of all *women having a daughter* can be represented by the concept term

$$\mathsf{Woman} \sqcap \exists \mathsf{child}.\mathsf{Woman},$$

and the concept of all *women having only daughters* by

$$\mathsf{Woman} \sqcap \forall \mathsf{child}.\mathsf{Woman}.$$

Knowledge representation systems based on DLs provide their users with various inference services that allow them to deduce implicit knowledge from the explicitly represented knowledge. For instance, the subsumption algorithm allows one to

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 350–364, 2009.
© Springer-Verlag Berlin Heidelberg 2009

determine subconcept-superconcept relationships. For example, the concept term Woman subsumes the concept term Woman $\sqcap \exists$child.Woman since all instances of the second term are also instances of the first term, i.e., the second term is always interpreted as a subset of the first term. With the help of the subsumption algorithm, a newly introduced concept term can automatically be placed at the correct position in the hierarchy of the already existing concept terms.

Two concept terms C, D are *equivalent* ($C \equiv D$) if they subsume each other, i.e., if they always represent the same set of individuals. For example, the terms $\forall$child.Rich $\sqcap \forall$child.Woman and $\forall$child.(Rich $\sqcap$ Woman) are equivalent since the value restriction operator ($\forall r.C$) distributes over the conjunction operator ($\sqcap$). If we replaced the value restriction operator by the existential restriction operator ($\exists r.C$), then this equivalence would no longer hold. However, for this operator, we still have the equivalence

$$\exists \text{child.Rich} \sqcap \exists \text{child.}(\text{Woman} \sqcap \text{Rich}) \equiv \exists \text{child.}(\text{Woman} \sqcap \text{Rich}).$$

The equivalence test can, for example, be used to find out whether a concept term representing a particular notion has already been introduced, thus avoiding multiple introduction of the same concept into the concept hierarchy. This inference capability is very important if the knowledge base containing the concept terms is very large, evolves during a long time period, and is extended and maintained by several knowledge engineers. However, testing for equivalence of concepts is not always sufficient to find out whether, for a given concept term, there already exists another concept term in the knowledge base describing the same notion. For example, assume that one knowledge engineer has defined the concept of all *women having a daughter* by the concept term

$$\text{Woman} \sqcap \exists \text{child.Woman}.$$

A second knowledge engineer might represent this notion in a somewhat more fine-grained way, e.g., by using the term Female $\sqcap$ Human in place of Woman. The concept terms Woman $\sqcap \exists$child.Woman and

$$\text{Female} \sqcap \text{Human} \sqcap \exists \text{child.}(\text{Female} \sqcap \text{Human})$$

are not equivalent, but they are meant to represent the same concept. The two terms can obviously be made equivalent by substituting the concept name Woman in the first term by the concept term Female $\sqcap$ Human. This leads us to *unification of concept terms*, i.e., the question whether two concept terms can be made equivalent by applying an appropriate substitution, where a substitution replaces (some of the) concept names by concept terms. Of course, it is not necessarily the case that unifiable concept terms are meant to represent the same notion. A unifiability test can, however, suggest to the knowledge engineer possible candidate terms.

Unification in DLs was first considered in [9] for a DL called $\mathcal{FL}_0$, which has the concept constructors *conjunction* ($\sqcap$), *value restriction* ($\forall r.C$), and the *top concept* ($\top$). It was shown that unification in $\mathcal{FL}_0$ is decidable and

ExpTime-complete, i.e., given an $\mathcal{FL}_0$-unification problem, we can effectively decide whether it has a solution or not, but in the worst-case, any such decision procedure needs exponential time. This result was extended in [7] to a more expressive DL, which additional has the role constructor *transitive closure*. Interestingly, the *unification type* of $\mathcal{FL}_0$ had been determined almost a decade earlier in [1]. In fact, as shown in [9], unification in $\mathcal{FL}_0$ corresponds to unification modulo the equational theory of idempotent Abelian monoids with several homomorphisms. In [1] it was shown that, already for a single homomorphism, unification modulo this theory has unification type zero, i.e., there are unification problems for this theory that do not have a minimal complete set of unifiers. In particular, such unification problems cannot have a finite complete set of unifiers.

In this paper, we consider unification in the DL $\mathcal{EL}$. The $\mathcal{EL}$-family consists of inexpressive DLs whose main distinguishing feature is that they provide their users with *existential restrictions* ($\exists r.C$) rather than value restrictions ($\forall r.C$) as the main concept constructor involving roles. The core language of this family is $\mathcal{EL}$, which has the top concept, conjunction, and existential restrictions as concept constructors. This family has recently drawn considerable attention since, on the one hand, the subsumption problem stays tractable (i.e., decidable in polynomial time) in situations where $\mathcal{FL}_0$, the corresponding DL with value restrictions, becomes intractable: subsumption between concept terms is tractable for both $\mathcal{FL}_0$ and $\mathcal{EL}$, but allowing the use of concept definitions or even more expressive terminological formalisms makes $\mathcal{FL}_0$ intractable [2,16,4], whereas it leaves $\mathcal{EL}$ tractable [3,13,4]. On the other hand, although of limited expressive power, $\mathcal{EL}$ is nevertheless used in applications, e.g., to define biomedical ontologies. For example, both the large medical ontology SNOMED CT[1] and the Gene Ontology[2] can be expressed in $\mathcal{EL}$, and the same is true for large parts of the medical ontology GALEN [18]. The importance of $\mathcal{EL}$ can also be seen from the fact that the new OWL 2 standard[3] contains a sub-profile OWL 2 EL, which is based on (an extension of) $\mathcal{EL}$.

Unification in $\mathcal{EL}$ has, to the best of our knowledge, not been investigated before, but matching (where one side of the equation(s) to be solved does not contain variables) has been considered in [6,17]. In particular, it was shown in [17] that the decision problem, i.e., the problem of deciding whether a given $\mathcal{EL}$-matching problem has a matcher or not, is NP-complete. Interestingly, $\mathcal{FL}_0$ behaves better w.r.t. matching than $\mathcal{EL}$: for $\mathcal{FL}_0$, the decision problem is tractable [8]. In this paper, we show that, w.r.t. the unification type, $\mathcal{FL}_0$ and $\mathcal{EL}$ behave the same: just as $\mathcal{FL}_0$, the DL $\mathcal{EL}$ has unification type zero. However, w.r.t. the decision problem, $\mathcal{EL}$ behaves much better than $\mathcal{FL}_0$: $\mathcal{EL}$-unification is NP-complete, and thus has the same complexity as $\mathcal{EL}$-matching.

In the next section, we define the DL $\mathcal{EL}$ and unification in $\mathcal{EL}$ more formally. In Section 3, we recall the characterisation of subsumption and equivalence in

[1] http://www.ihtsdo.org/snomed-ct/
[2] http://www.geneontology.org/
[3] See http://www.w3.org/TR/owl2-profiles/

$\mathcal{EL}$ from [17], and in Section 4 we use this to show that unification in $\mathcal{EL}$ has type zero. In Section 5, we show that unification in $\mathcal{EL}$ is NP-complete, and in Section 6 we point out that our results for $\mathcal{EL}$-unification imply that unification modulo the equational theory of semilattices with monotone operators [19] is NP-complete and of unification type zero.

More information about Description Logics can be found in [5], and about unification theory in [12].

2 Unification in $\mathcal{EL}$

First, we define the syntax and semantics of $\mathcal{EL}$-concept terms as well as the subsumption and the equivalence relation on these terms.

Starting with a set N_{con} of concept names and a set N_{role} of role names, *$\mathcal{EL}$-concept terms* are built using the concept constructors top concept ($\top$), conjunction ($\sqcap$), and existential restriction ($\exists r.C$). The semantics of $\mathcal{EL}$ is defined in the usual way, using the notion of an interpretation $\mathcal{I} = (\mathcal{D}_\mathcal{I}, \cdot^\mathcal{I})$, which consists of a nonempty domain $\mathcal{D}_\mathcal{I}$ and an interpretation function $\cdot^\mathcal{I}$ that assigns binary relations on $\mathcal{D}_\mathcal{I}$ to role names and subsets of $\mathcal{D}_\mathcal{I}$ to concept terms, as shown in the semantics column of Table 1.

Table 1. Syntax and semantics of $\mathcal{EL}$

Name	Syntax	Semantics
concept name	A	$A^\mathcal{I} \subseteq \mathcal{D}_\mathcal{I}$
role name	r	$r^\mathcal{I} \subseteq \mathcal{D}_\mathcal{I} \times \mathcal{D}_\mathcal{I}$
top-concept	$\top$	$\top^\mathcal{I} = \mathcal{D}_\mathcal{I}$
conjunction	$C \sqcap D$	$(C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$
existential restriction	$\exists r.C$	$(\exists r.C)^\mathcal{I} = \{x \mid \exists y : (x, y) \in r^\mathcal{I} \wedge y \in C^\mathcal{I}\}$
subsumption	$C \sqsubseteq D$	$C^\mathcal{I} \subseteq D^\mathcal{I}$
equivalence	$C \equiv D$	$C^\mathcal{I} = D^\mathcal{I}$

The concept term C *is subsumed by* the concept term D (written $C \sqsubseteq D$) iff $C^\mathcal{I} \subseteq D^\mathcal{I}$ holds for all interpretations $\mathcal{I}$. We say that C *is equivalent to* D (written $C \equiv D$) iff $C \sqsubseteq D$ and $D \sqsubseteq C$, i.e., iff $C^\mathcal{I} = D^\mathcal{I}$ holds for all interpretations $\mathcal{I}$. The concept term C *is strictly subsumed by* the concept term D (written $C \sqsubset D$) iff $C \sqsubseteq D$ and $C \not\equiv D$.

A *concept definition* is of the form $A \doteq C$ where A is a concept name and C is a concept term. A *TBox* $\mathcal{T}$ is a finite set of concept definitions such that no concept name occurs more than once on the left-hand side of a concept definition in $\mathcal{T}$. The TBox $\mathcal{T}$ is called *acyclic* if there are no cyclic dependencies between its concept definitions. The interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$ iff $A^\mathcal{I} = C^\mathcal{I}$ holds for all concept definitions $A \doteq C$ in $\mathcal{T}$. Subsumption and equivalence w.r.t. a TBox are defined as follows: $C \sqsubseteq_\mathcal{T} D$ ($C \equiv_\mathcal{T} D$) iff $C^\mathcal{I} \subseteq D^\mathcal{I}$ ($C^\mathcal{I} = D^\mathcal{I}$) holds for all models $\mathcal{I}$ of $\mathcal{T}$. Subsumption and equivalence

w.r.t. an acyclic TBox can be reduced to subsumption and equivalence of concept terms (without TBox) by *expanding* the concept terms w.r.t. the TBox, i.e., by replacing defined concepts (i.e., concept names occurring on the left-hand side of a definition) by their definitions (i.e., the corresponding right-hand sides) until all defined concepts have been replaced. This expansion process may, however, result in an exponential blow-up [10].

In order to define unification of concept terms, we first introduce the notion of a substitution operating on concept terms. To this purpose, we partition the set of concepts names into a set N_v of concept variables (which may be replaced by substitutions) and a set N_c of concept constants (which must not be replaced by substitutions). Intuitively, N_v are the concept names that have possibly been given another name or been specified in more detail in another concept term describing the same notion. The elements of N_c are the ones of which it is assumed that the same name is used by all knowledge engineers (e.g., standardised names in a certain domain).

A *substitution* σ is a mapping from N_v into the set of all $\mathcal{EL}$-concept terms. This mapping is extended to concept terms in the obvious way, i.e.,

- $\sigma(A) := A$ for all $A \in N_c$,
- $\sigma(\top) := \top$,
- $\sigma(C \sqcap D) := \sigma(C) \sqcap \sigma(D)$, and
- $\sigma(\exists r.C) := \exists r.\sigma(C)$.

Definition 1. *An* $\mathcal{EL}$-unification problem *is of the form* $\Gamma = \{C_1 \equiv^? D_1, \ldots, C_n \equiv^? D_n\}$, *where* $C_1, D_1, \ldots C_n, D_n$ *are* $\mathcal{EL}$*-concept terms. The substitution* σ *is a* unifier *(or* solution*) of* Γ *iff* $\sigma(C_i) \equiv \sigma(D_i)$ *for* $i = 1, \ldots, n$. *In this case,* Γ *is called* solvable *or* unifiable.

When we say that $\mathcal{EL}$-unification is *decidable (NP-complete)*, then we mean that the following decision problem is decidable (NP-complete): given an $\mathcal{EL}$-unification problem Γ, decide whether Γ is solvable or not.

As usual, unifiers can be compared using the instantiation preorder $\preceq$. Let Γ be an $\mathcal{EL}$-unification problem, V the set of variables occurring in Γ, and σ, θ two unifiers of this problem. We define

$$\sigma \preceq \theta \text{ iff there is a substitution } \lambda \text{ such that } \theta(X) \equiv \lambda(\sigma(X)) \text{ for all } X \in V.$$

If $\sigma \preceq \theta$, then we say that θ is an *instance* of σ.

Definition 2. *Let* Γ *be an* $\mathcal{EL}$*-unification problem. The set of substitutions* M *is called a* complete set of unifiers *for* Γ *iff it satisfies the following two properties:*

1. *every element of* M *is a unifier of* Γ*;*
2. *if* θ *is a unifier of* Γ*, then there exists a unifier* $\sigma \in M$ *such that* $\sigma \preceq \theta$.

The set M *is called a* minimal complete set of unifiers *for* Γ *iff it additionally satisfies*

3. *if* $\sigma, \theta \in M$*, then* $\sigma \preceq \theta$ *implies* $\sigma = \theta$.

The unification type of a given unification problem is determined by the existence and cardinality of such a minimal complete set.

Definition 3. *Let Γ be an $\mathcal{EL}$-unification problem. This problem has type* unitary *(*finitary*,* infinitary*) iff it has a minimal complete set of unifiers of cardinality* 1 *(finite cardinality, infinite cardinality). If Γ does not have a minimal complete set of unifiers, then it is of type* zero*.*

Note that the set of all unifiers of a given $\mathcal{EL}$-unification problem is always a complete set of unifiers. However, this set is usually infinite and redundant (in the sense that some unifiers are instances of others). For a unitary or finitary $\mathcal{EL}$-unification problem, all unifiers can be represented by a finite complete set of unifiers, whereas for problems of type infinitary or zero this is no longer possible. In fact, if a problem has a finite complete set of unifiers M, then it also has a finite *minimal* complete set of unifiers, which can be obtained by iteratively removing redundant elements from M. For an infinite complete set of unifiers, this approach of removing redundant unifiers may be infinite, and the set reached in the limit need no longer be complete. This is what happens for problems of type zero. The difference between infinitary and type zero is that a unification problem of type zero cannot even have a non-redundant complete set of unifiers, i.e., every complete set of unifiers must contain different unifiers σ, θ such that $\sigma \not\preceq \theta$.

When we say that $\mathcal{EL}$ *has unification type zero*, we mean that there exists an $\mathcal{EL}$-unification problem that has type zero. Before we can prove that this is indeed the case, we must first have a closer look at equivalence in $\mathcal{EL}$.

3 Equivalence and Subsumption in $\mathcal{EL}$

In order to characterise equivalence of $\mathcal{EL}$-concept terms, the notion of a reduced $\mathcal{EL}$-concept term is introduced in [17]. A given $\mathcal{EL}$-concept term can be transformed into an equivalent reduced term by applying the following rules modulo associativity and commutativity of conjunction:

$C \sqcap \top \to C$	for all $\mathcal{EL}$-concept terms C
$A \sqcap A \to A$	for all concept names $A \in N_{con}$
$\exists r.C \sqcap \exists r.D \to \exists r.C$	for all $\mathcal{EL}$-concept terms C, D with $C \sqsubseteq D$

Obviously, these rules are equivalence preserving. We say that the $\mathcal{EL}$-concept term C is *reduced* if none of the above rules is applicable to it (modulo associativity and commutativity of $\sqcap$). The $\mathcal{EL}$-concept term D is a *reduced form* of C if D is reduced and can be obtained from C by applying the above rules (modulo associativity and commutativity of $\sqcap$). The following theorem is an easy consequence of Theorem 6.3.1 on page 181 of [17].

Theorem 1. *Let C, D be $\mathcal{EL}$-concept terms, and $\widehat{C}, \widehat{D}$ reduced forms of C, D, respectively. Then $C \equiv D$ iff $\widehat{C}$ is identical to $\widehat{D}$ up to associativity and commutativity of $\sqcap$.*

This theorem can also be used to derive a recursive characterisation of subsumption in $\mathcal{EL}$. In fact, if $C \sqsubseteq D$, then $C \sqcap D \equiv C$, and thus C and $C \sqcap D$ have the same reduced form. Thus, during reduction, all concept names and existential restrictions of D must be "eaten up" by corresponding concept names and existential restrictions of C.

Corollary 1. *Let $C = A_1 \sqcap \ldots \sqcap A_k \sqcap \exists r_1.C_1 \sqcap \ldots \sqcap \exists r_m.C_m$ and $D = B_1 \sqcap \ldots \sqcap B_\ell \sqcap \exists s_1.D_1 \sqcap \ldots \sqcap \exists s_n.D_n$, where $A_1, \ldots, A_k, B_1, \ldots, B_\ell$ are concept names. Then $C \sqsubseteq D$ iff $\{B_1, \ldots, B_\ell\} \subseteq \{A_1, \ldots, A_k\}$ and for every $j, 1 \leq j \leq n$, there exists an $i, 1 \leq i \leq m$, such that $r_i = s_j$ and $C_i \sqsubseteq D_j$.*

Note that this corollary also covers the cases where some of the numbers k, ℓ, m, n are zero. The empty conjunction should then be read as $\top$. The following lemma, which is an immediate consequence of this corollary, will be used in our proof that $\mathcal{EL}$ has unification type zero.

Lemma 1. *If C, D are reduced $\mathcal{EL}$-concept terms such that $\exists r.D \sqsubseteq C$, then C is either $\top$, or of the form $C = \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n$ where $n \geq 1$; $C_1, \ldots, C_n$ are reduced and pairwise incomparable w.r.t. subsumption; and $D \sqsubseteq C_1, \ldots, D \sqsubseteq C_n$. Conversely, if C, D are $\mathcal{EL}$-concept terms such that $C = \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n$ and $D \sqsubseteq C_1, \ldots, D \sqsubseteq C_n$, then $\exists r.D \sqsubseteq C$.*

In the proof of decidability of $\mathcal{EL}$-unification, we will make use of the fact that the inverse strict subsumption order is well-founded.

Proposition 1. *There is no infinite sequence $C_0, C_1, C_2, C_3, \ldots$ of $\mathcal{EL}$-concept terms such that $C_0 \sqsubset C_1 \sqsubset C_2 \sqsubset C_3 \sqsubset \cdots$.*

Proof. We define the *role depth* of an $\mathcal{EL}$-concept term C as the maximal nesting of existential restrictions in C. Let n_0 be the role depth of C_0. Since $C_0 \sqsubseteq C_i$ for $i \geq 1$, it is an easy consequence of Corollary 1 that the role depth of C_i is bounded by n_0, and that C_i contains only concept and role names occurring in C_0. In addition, it is known that, for a given natural number n_0 and finite sets of concept names $\mathcal{C}$ and role names $\mathcal{R}$, there are, up to equivalence, only finitely many $\mathcal{EL}$-concept term built using concept names from $\mathcal{C}$ and role names from $\mathcal{R}$ and of a role depth bounded by n_0 [11]. Consequently, there are indices $i < j$ such that $C_i \equiv C_j$. This contradicts our assumption that $C_i \sqsubset C_j$. □

4 An $\mathcal{EL}$-Unification Problem of Type Zero

To show that $\mathcal{EL}$ has unification type zero, we exhibit an $\mathcal{EL}$-unification problem that has this type.

Theorem 2. *Let X, Y be variables. The $\mathcal{EL}$-unification problem $\Gamma := \{X \sqcap \exists r.Y \equiv^? \exists r.Y\}$ has unification type zero.*

Proof. It is enough to show that any complete set of unifiers for this problem is redundant, i.e., contains two different unifiers that are comparable w.r.t. the instantiation preorder. Thus, let M be a complete set of unifiers for Γ.

First, note that M must contain a unifier that maps X to an $\mathcal{EL}$-concept term not equivalent to $\top$ or $\exists r.\top$. In fact, consider a substitution τ such that $\tau(X) = \exists r.A$ and $\tau(Y) = A$. Obviously, τ is a unifier of Γ. Thus, M must contain a unifier σ such that $\sigma \preceq \tau$. In particular, this means that there is a substitution λ such that $\exists r.A = \tau(X) \equiv \lambda(\sigma(X))$. Obviously, $\sigma(X) \equiv \top$ ($\sigma(X) \equiv \exists r.\top$) would imply $\lambda(\sigma(X)) \equiv \top$ ($\lambda(\sigma(X)) \equiv \exists r.\top$), and thus $\exists r.A \equiv \top$ ($\exists r.A \equiv \exists r.\top$), which is, however, not the case.

Thus, let $\sigma \in M$ be such that $\sigma(X) \not\equiv \top$ and $\sigma(X) \not\equiv \exists r.\top$. Without loss of generality, we assume that $C := \sigma(X)$ and $D := \sigma(Y)$ are reduced. Since σ is a unifier of Γ, we have $\exists r.D \sqsubseteq C$. Consequently, Lemma 1 yields that C is of the form $C = \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n$ where $n \geq 1$, $C_1, \ldots, C_n$ are reduced and pairwise incomparable w.r.t. subsumption, and $D \sqsubseteq C_1, \ldots, D \sqsubseteq C_n$.

We use σ to construct a new unifier $\widehat{\sigma}$ as follows:

$$\begin{aligned} \widehat{\sigma}(X) &:= \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n \sqcap \exists r.Z \\ \widehat{\sigma}(Y) &:= D \sqcap Z \end{aligned}$$

where Z is a new variable (i.e., one not occurring in C, D). The second part of Lemma 1 implies that $\widehat{\sigma}$ is indeed a unifier of Γ.

Next, we show that $\widehat{\sigma} \preceq \sigma$. To this purpose, we consider the substitution λ that maps Z to C_1, and does not change any of the other variables. Then we have $\lambda(\widehat{\sigma}(X)) = \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n \sqcap \exists r.C_1 \equiv \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n = \sigma(X)$ and $\lambda(\widehat{\sigma}(Y)) = D \sqcap C_1 \equiv D = \sigma(Y)$. Note that the second equivalence holds since we have $D \sqsubseteq C_1$.

Since M is complete, there exists a unifier $\theta \in M$ such that $\theta \preceq \widehat{\sigma}$. Transitivity of the relation $\preceq$ thus yields $\theta \preceq \sigma$. Since σ and θ both belong to M, we have completed the proof of the theorem once we have shown that $\sigma \neq \theta$. Assume to the contrary that $\sigma = \theta$. Then we have $\sigma \preceq \widehat{\sigma}$, and thus there exists a substitution μ such that $\mu(\sigma(X)) \equiv \widehat{\sigma}(X)$, i.e.,

$$\exists r.\mu(C_1) \sqcap \ldots \sqcap \exists r.\mu(C_n) \equiv \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n \sqcap \exists r.Z. \tag{1}$$

Recall that the concept terms $C_1, \ldots, C_n$ are reduced and pairwise incomparable w.r.t. subsumption. In addition, since $\sigma(X) = \exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n$ is reduced and not equivalent to $\exists r.\top$, none of the concept terms $C_1, \ldots, C_n$ can be equivalent to $\top$. Finally, Z is a concept name that does not occur in $C_1, \ldots, C_n$. All this implies that $\exists r.C_1 \sqcap \ldots \sqcap \exists r.C_n \sqcap \exists r.Z$ is reduced. Obviously, any reduced form for $\exists r.\mu(C_1) \sqcap \ldots \sqcap \exists r.\mu(C_n)$ is a conjunction of at most n existential restrictions. Thus, Theorem 1 shows that the above equivalence (1) actually cannot hold.

To sum up, we have shown that M contains two distinct unifiers σ, θ such that $\theta \preceq \sigma$. Since M was an arbitrary complete set of unifiers for Γ, this shows that this unification problem cannot have a minimal complete set of unifiers. □

5 The Decision Problem

Before we can describe our decision procedure for $\mathcal{EL}$-unification, we must introduce some notation. An $\mathcal{EL}$-concept term is called an *atom* iff it is a concept name (i.e., concept constant or concept variable) or an existential restriction $\exists r.D$. Obviously, any $\mathcal{EL}$-concept term is (equivalent to) a conjunction of atoms, where the empty conjunction is $\top$. The set $At(C)$ of *atoms of an $\mathcal{EL}$-concept term* C is defined inductively: if $C = \top$, then $At(C) := \emptyset$; if C is a concept name, then $At(C) := \{C\}$; if $C = \exists r.D$ then $At(C) := \{C\} \cup At(D)$; if $C = C_1 \sqcap C_2$, then $At(C) := At(C_1) \cup At(C_2)$.

Concept names and existential restrictions $\exists r.D$ where D is a concept name or $\top$ are called *flat atoms*. The $\mathcal{EL}$-unification problem Γ is *flat* iff it only contains equations of the following form:

- $X \equiv^? C$ where X is a variable and C is a non-variable flat atom;
- $X_1 \sqcap \ldots \sqcap X_m \equiv^? Y_1 \sqcap \ldots \sqcap Y_n$ where $X_1, \ldots, X_m, Y_1, \ldots, Y_n$ are variables.

By introducing new concept variables and eliminating $\top$, any $\mathcal{EL}$-unification problem Γ can be transformed in polynomial time into a flat $\mathcal{EL}$-unification problem Γ' such that Γ is solvable iff Γ' is solvable. Thus, we may assume without loss of generality that our input $\mathcal{EL}$-unification problems are flat. Given a flat $\mathcal{EL}$-unification problem $\Gamma = \{C_1 \equiv^? D_1, \ldots, C_n \equiv^? D_n\}$, we call the atoms of $C_1, D_1, \ldots, C_n, D_n$ the *atoms of* Γ.

The unifier σ of Γ is called *reduced* (*ground*) iff, for all concept variables X occurring in Γ, the $\mathcal{EL}$-concept term $\sigma(X)$ is reduced (does not contain variables). Obviously, Γ is solvable iff it has a reduced ground unifier. Given a ground unifier σ of Γ, we consider the set $At(\sigma)$ of all atoms of $\sigma(X)$, where X ranges over all variables occurring in Γ. We call the elements of $At(\sigma)$ the *atoms of* σ.

Given $\mathcal{EL}$-concept terms C, D, we define $C >_{is} D$ iff $C \sqsubset D$. Proposition 1 says that the strict order $>_{is}$ defined this way is well-founded. This order is monotone in the following sense.

Lemma 2. *Let C, D, D' be $\mathcal{EL}$-concept terms such that $D >_{is} D'$ and C is reduced and contains at least one occurrence of D. If C' is obtained from C by replacing all occurrences of D by D', then $C >_{is} C'$.*

Proof. We prove the lemma by induction on the size of C. If $C = D$, then $C' = D'$, and thus $C = D >_{is} D' = C'$. Thus, assume that $C \neq D$. In this case, C obviously cannot be a concept name. If $C = \exists r.C_1$, then D occurs in C_1. By induction, we can assume that $C_1 >_{is} C_1'$, where C_1' is obtained from C_1 by replacing all occurrences of D by D'. Thus, we have $C = \exists r.C_1 >_{is} \exists r.C_1' = C'$ by Corollary 1. Finally, assume that $C = C_1 \sqcap \ldots \sqcap C_n$ for $n > 1$ atoms $C_1, \ldots, C_n$. Since C is reduced, these atoms are incomparable w.r.t. subsumption, and since D occurs in C we can assume without loss of generality that D occurs in C_1. Let $C_1', \ldots, C_n'$ be respectively obtained from $C_1, \ldots, C_n$ by replacing every occurrence of D by D', and then reducing the concept term obtained this way. By induction, we have $C_1 >_{is} C_1'$. Assume that $C \not>_{is} C'$.

Since the concept constructors of $\mathcal{EL}$ are monotone w.r.t. subsumption $\sqsubseteq$, we have $C \sqsubseteq C'$, and thus $C \not>_{is} C'$ means that $C \equiv C'$. Consequently, $C = C_1 \sqcap \ldots \sqcap C_n$ and the reduced form of $C'_1 \sqcap \ldots \sqcap C'_n$ must be equal up to associativity and commutativity of $\sqcap$. If $C'_1 \sqcap \ldots \sqcap C'_n$ is not reduced, then its reduced form is actually a conjunction of $m < n$ atoms, which contradicts $C \equiv C'$. If $C'_1 \sqcap \ldots \sqcap C'_n$ is reduced, then $C_1 >_{is} C'_1$ implies that there is an $i \neq 1$ such that $C_i \equiv C'_1$. However, then $C_i \equiv C'_1 \sqsupseteq C_1$ contradicts the fact that the atoms $C_1, \ldots, C_n$ are incomparable w.r.t. subsumption. □

We use the order $>_{is}$ on $\mathcal{EL}$-concept terms to define a well-founded order on ground unifiers. Since $>_{is}$ is well-founded, its multiset extension $>_m$ is also well-founded. Given a ground unifier σ of Γ, we consider the multiset $S(\sigma)$ of all $\mathcal{EL}$-concept terms $\sigma(X)$, where X ranges over all concept variables occurring in Γ. For two ground unifiers σ, θ of Γ, we define $\sigma \succ \theta$ iff $S(\sigma) >_m S(\theta)$. The ground unifier σ of Γ is *minimal* iff there is no ground unifier θ of Γ such that $\sigma \succ \theta$. The following proposition is an easy consequence of the fact that $\succ$ is well-founded.

Proposition 2. *Let Γ be an $\mathcal{EL}$-unification problem. Then Γ is solvable iff it has a minimal reduced ground unifier.*

In the following, we show that minimal reduced ground unifiers of flat $\mathcal{EL}$-unification problems satisfy properties that make it easy to check (with an NP-algorithm) whether such a unifier exists or not.

Lemma 3. *Let Γ be a flat $\mathcal{EL}$-unification problem and γ a minimal reduced ground unifier of Γ. If C is an atom of γ, then there is a non-variable atom D of Γ such that $C \equiv \gamma(D)$.*

Proof. Since γ is ground, C is either a concept constant or an existential restriction. First, assume that $C = A$ for a concept constant A, but there is no non-variable atom D of Γ such that $A \equiv \gamma(D)$. This simply means that A does not occur in Γ. Let γ' be the substitution obtained from γ by replacing every occurrence of A by $\top$. Since equivalence in $\mathcal{EL}$ is preserved under replacing concept names by $\top$, and since A does not occur in Γ, it is easy to see that γ' is also a unifier of Γ. However, since $\gamma \succ \gamma'$, this contradicts our assumption that γ is minimal.

Second, assume that $C = \exists r.C_1$, but there is no non-variable atom D of Γ such that $C \equiv \gamma(D)$. We assume that C is maximal (w.r.t. subsumption) with this property, i.e., for every atom C' of γ with $C \sqsubset C'$, there is a non-variable atom D' of Γ such that $C' \equiv \gamma(D')$. Let $D_1, \ldots, D_n$ be all the atoms of Γ with $C \sqsubseteq \gamma(D_i)$ $(i = 1, \ldots, n)$. By our assumptions on C, we actually have $C \sqsubset \gamma(D_i)$ and, by Lemma 1, the atom D_i is also an existential restriction $D_i = \exists r.D'_i$ $(i = 1, \ldots, n)$. The conjunction $\widehat{D} := \gamma(D_1) \sqcap \ldots \sqcap \gamma(D_n)$ obviously subsumes C. We claim that this subsumption relationship is actually strict. In fact, if $n = 0$, then $\widehat{D} = \top$, and since C is an atom, it is not equivalent to $\top$. If $n \geq 1$, then $C = \exists r.C_1 \sqsupseteq \exists r.\gamma(D'_1) \sqcap \ldots \sqcap \exists r.\gamma(D_n)$ would imply (by

Corollary 1) that there is an $i, 1 \leq i \leq n$, with $C_1 \sqsupseteq \gamma(D'_i)$. However, this would yield $C = \exists r.C_1 \sqsupseteq \exists r.\gamma(D'_i) = \gamma(D_i)$, which contradicts the fact that $C \sqsubset \gamma(D_i)$. Thus, we have shown that $C \sqsubset \widehat{D}$. The substitution γ' is obtained from γ by replacing every occurrence of C by $\widehat{D}$. Lemma 2 implies that $\gamma \succ \gamma'$. Thus, to obtain the desired contradiction, it is sufficient to show that γ' is a unifier of Γ.

First, consider an equation of the form $X \equiv^? E$ in Γ, where X is a variable and E is a non-variable flat atom. If E is a concept constant, then $\gamma(X) = E$, and thus $\gamma'(X) = \gamma(X)$, which shows that γ' solves this equation. Thus, assume that $E = \exists r.E'$. Since γ is reduced, we actually have $\gamma(X) = \exists r.\gamma(E')$. If C occurs in $\gamma(E')$, then each replacement of C by $\widehat{D}$ in $\gamma(E')$ is matched by the corresponding replacement in $\gamma(X)$. Thus, in this case γ' again solves the equation. Finally, assume that $C = \gamma(X)$. But then $C \equiv \gamma(E)$ for a non-variable atom E of Γ, which contradicts our assumption on C.

Second, consider an equation of the form $X_1 \sqcap \ldots \sqcap X_m \equiv^? Y_1 \sqcap \ldots \sqcap Y_n$ where $X_1, \ldots, X_m, Y_1, \ldots, Y_n$ are variables. Then $L := \gamma(X_1 \sqcap \ldots \sqcap X_m)$ and $R := \gamma(Y_1 \sqcap \ldots \sqcap Y_n)$ reduce to the same reduced $\mathcal{EL}$-concept term J. Let L', R', J' be the $\mathcal{EL}$-concept terms respectively obtained from L, R, J by replacing every occurrence of C by $\widehat{D}$. We prove that $L' = \gamma'(X_1 \sqcap \ldots \sqcap X_m)$ and $R' = \gamma'(Y_1 \sqcap \ldots \sqcap Y_n)$ both reduce to J', which shows that γ' solves this equation. It is enough to show that the reductions are invariant under the replacement of C by $\widehat{D}$. Obviously, all the interesting reductions are of the form $E_1 \sqcap E_2 \to E_1$ where E_1, E_2 are existential restrictions such that $E_1 \sqsubseteq E_2$. Since γ is reduced, we can assume that E_1, E_2 are reduced. Let E'_1, E'_2 be respectively obtained from E_1, E_2 by replacing every occurrence of C by $\widehat{D}$. We must show that $E'_1 \sqcap E'_2$ reduces to E'_1. For this, it is enough to show that $E'_1 \sqsubseteq E'_2$. Assume that an occurrence of C in E_1 is actually needed to have the subsumption $E_1 \sqsubseteq E_2$. Then there is an existential restriction C' in E_2 such that $C \sqsubseteq C'$. If $C = C'$, then both are replaced by $\widehat{D}$, and thus this replacement is harmless. Otherwise, $C \sqsubset C'$. Since C' is an atom of γ, maximality of C yields that there is a non-variable atom D' of Γ such that $C' \equiv \gamma(D')$. Now $C \sqsubset C' \equiv \gamma(D')$ implies that there is an $i, 1 \leq i \leq n$, such that $D' = D_i$. Thus, C' is actually one of the conjuncts of $\widehat{D}$, which again shows that replacing C by $\widehat{D}$ is harmless. Thus, we have shown that $E'_1 \sqsubseteq E'_2$, which completes the proof of the lemma. □

The next proposition is an easy consequence of this lemma.

Proposition 3. *Let Γ be a flat $\mathcal{EL}$-unification problem and γ a minimal reduced ground unifier of Γ. If X is a concept variable occurring in Γ, then $\gamma(X) \equiv \top$ or there are non-variable atoms $D_1, \ldots, D_n$ ($n \geq 1$) of Γ such that $\gamma(X) \equiv \gamma(D_1) \sqcap \ldots \sqcap \gamma(D_n)$.*

Proof. If $\gamma(X) \not\equiv \top$, then it is a non-empty conjunction of atoms, i.e., there are atoms $C_1, \ldots, C_n$ ($n \geq 1$) such that $\gamma(X) = C_1 \sqcap \ldots \sqcap C_n$. Then $C_1, \ldots, C_n$ are atoms of γ, and thus Lemma 3 yields non-variable atoms $D_1, \ldots, D_n$ of Γ such that $C_i \equiv \gamma(D_i)$ for $i = 1, \ldots n$. Consequently, $\gamma(X) \equiv \gamma(D_1) \sqcap \ldots \sqcap \gamma(D_n)$. □

This proposition suggests the following *non-deterministic algorithm for deciding solvability of a given flat $\mathcal{EL}$-unification problem Γ:*

1. For every variable X occurring in Γ, guess a finite, possibly empty, set S_X of non-variable atoms of Γ.
2. We say that the variable X *directly depends on* the variable Y if Y occurs in an atom of S_X. Let *depends on* be the transitive closure of *directly depends on*. If there is a variable that depends on itself, then the algorithm returns "fail." Otherwise, there exists a strict linear order $>$ on the variables occurring in Γ such that $X > Y$ if X depends on Y.
3. We define the substitution σ along the linear order $>$:
   - If X is the least variable w.r.t. $>$, then S_X does not contain any variables. We define $\sigma(X)$ to be the conjunction of the elements of S_X, where the empty conjunction is $\top$.
   - Assume that $\sigma(Y)$ is defined for all variables $Y < X$. Then S_X only contains variables Y for which $\sigma(Y)$ is already defined. If S_X is empty, then we define $\sigma(X) := \top$. Otherwise, let $S_X = \{D_1, \ldots, D_n\}$. We define $\sigma(X) := \sigma(D_1) \sqcap \ldots \sqcap \sigma(D_n)$.
4. Test whether the substitution σ computed in the previous step is a unifier of Γ. If this is the case, then return σ; otherwise, return "fail."

This algorithm is trivially *sound* since it only returns substitutions that are unifiers of Γ. In addition, it obviously always terminates. Thus, to show correctness of our algorithm, it is sufficient to show that it is complete.

Lemma 4 (completeness). *If Γ is solvable, then there is a way of guessing in Step 1 subsets S_X of the non-variable atoms of Γ such that the* depends on *relation determined in Step 2 is acyclic and the substitution σ computed in Step 3 is a unifier of Γ.*

Proof. If Γ is solvable, then it has a minimal reduced ground unifier γ. By Proposition 3, for every variable X occurring in Γ we have $\gamma(X) \equiv \top$ or there are non-variable atoms $D_1, \ldots, D_n$ $(n \geq 1)$ of Γ such that $\gamma(X) \equiv \gamma(D_1) \sqcap \ldots \sqcap \gamma(D_n)$. If $\gamma(X) \equiv \top$, then we define $S_X := \emptyset$. Otherwise, we define $S_X := \{D_1, \ldots, D_n\}$.

We show that the relation *depends on* induced by these sets S_X is acyclic, i.e., there is no variable X such that X depends on itself. If X directly depends on Y, then Y occurs in an element of S_X. Since S_X consists of non-variable atoms of the flat unification problem Γ, this means that there is a role name r such that $\exists r.Y \in S_X$. Consequently, we have $\gamma(X) \sqsubseteq \exists r.\gamma(Y)$. Thus, if X depends on X, then there are $k \geq 1$ role names $r_1, \ldots, r_k$ such that $\gamma(X) \sqsubseteq \exists r_1. \cdots \exists r_k.\gamma(X)$. This is clearly not possible since $\gamma(X)$ cannot be subsumed by an $\mathcal{EL}$-concept term whose role depth is larger than the role depth of $\gamma(X)$.

To show that the substitution σ induced by the sets S_X is a unifier of Γ, we prove that σ is equivalent to γ, i.e., $\sigma(X) \equiv \gamma(X)$ holds for all variables X occurring in Γ. The substitution σ is defined along the linear order $>$. If X is the least variable w.r.t. $>$, then S_X does not contain any variables. If S_X is empty, then $\sigma(X) = \top \equiv \gamma(X)$. Otherwise, let $S_X = \{D_1, \ldots, D_n\}$. Since the atoms D_i do not contain variables, we have $D_i = \gamma(D_i)$. Thus, the definitions of S_X and of σ yield $\sigma(X) = D_1 \sqcap \ldots \sqcap D_n = \gamma(D_1) \sqcap \ldots \sqcap \gamma(D_n) \equiv \gamma(X)$.

Assume that $\sigma(Y) \equiv \gamma(Y)$ holds for all variables $Y < X$. If $S_X = \emptyset$, then we have again $\sigma(X) = \top \equiv \gamma(X)$. Otherwise, let $S_X = \{D_1, \ldots, D_n\}$. Since the atoms D_i contain only variables that are smaller than X, we have $\sigma(D_i) \equiv \gamma(D_i)$ by induction. Thus, the definitions of S_X and of σ yield $\sigma(X) = \sigma(D_1) \sqcap \ldots \sqcap \sigma(D_n) \equiv \gamma(D_1) \sqcap \ldots \sqcap \gamma(D_n) \equiv \gamma(X)$. □

Note that our proof of completeness actually shows that, up to equivalence, the algorithm returns all minimal reduced ground unifiers of Γ.

Theorem 3. *$\mathcal{EL}$-unification is NP-complete.*

Proof. NP-hardness follows from the fact that $\mathcal{EL}$-matching is NP-complete [17]. To show that the problem can be decided by a non-deterministic polynomial-time algorithm, we analyse the complexity of our algorithm. Obviously, guessing the sets S_X (Step 1) can be done within NP. Computing the *depends on* relation and checking it for acyclicity (Step 2) is clearly polynomial.

Steps 3 and 4 are more problematic. In fact, since a variable may occur in different atoms of Γ, the substitution σ computed in Step 3 may be of exponential size. This is actually the same reason that makes a naive algorithm for syntactic unification compute an exponentially large most general unifier [12]. As in the case of syntactic unification, the solution to this problem is basically structure sharing. Instead of computing the substitution σ explicitly, we view its definition as an acyclic TBox. To be more precise, for every concept variable X occurring in Γ, the TBox $\mathcal{T}_\sigma$ contains the concept definition $X \doteq \top$ if $S_X = \emptyset$ and $X \doteq D_1 \sqcap \ldots \sqcap D_n$ if $S_X = \{D_1, \ldots, D_n\}$ $(n \geq 1)$. Instead of computing σ in Step 3, we compute $\mathcal{T}_\sigma$. Because of the acyclicity test in Step 2, we know that $\mathcal{T}_\sigma$ is an acyclic TBox. The size of $\mathcal{T}_\sigma$ is obviously polynomial in the size of Γ, and thus this modified Step 3 is polynomial. It is easy to see that applying the substitution σ is the same as expanding the concept terms C, D w.r.t. the TBox $\mathcal{T}_\sigma$. This implies that, for every equation $C \equiv^? D$ in Γ, we have $C \equiv_{\mathcal{T}_\sigma} D$ iff $\sigma(C) \equiv \sigma(D)$. Thus, testing whether σ is a unifier of Γ can be reduced to testing whether $C \equiv_{\mathcal{T}_\sigma} D$ holds for every equation $C \equiv^? D$ in Γ. Since subsumption (and thus equivalence) in $\mathcal{EL}$ w.r.t. acyclic TBoxes can be decided in polynomial time [3],[4] this completes the proof of the theorem. □

6 Unification in Semilattices with Monotone Operators

Unification problems and their types were originally not introduced for Description Logics, but for equational theories [12]. In this section, we show that the above results for unification in $\mathcal{EL}$ can actually be viewed as results for an equational theory. As shown in [19], the equivalence problem for $\mathcal{EL}$-concept terms corresponds to the word problem for the equational theory of semilattices with monotone operators. In order to define this theory, we consider a signature Σ_{SLmO} consisting of a binary function symbol $\wedge$, a constant symbol 1, and finitely many unary function symbols $f_1, \ldots, f_n$. Terms can then be built using these symbols and additional variable symbols and free constant symbols.

[4] Of course, the polynomial-time subsumption algorithm does not expand the TBox.

Definition 4. *The equational theory of* semilattices with monotone operators *is defined by the following identities:*

$$SLmO := \{x \wedge (y \wedge z) = (x \wedge y) \wedge z,\ x \wedge y = y \wedge x,\ x \wedge x = x,\ x \wedge 1 = x\} \cup$$
$$\{f_i(x \wedge y) \wedge f_i(y) = f_i(x \wedge y) \mid 1 \leq i \leq n\}$$

A given $\mathcal{EL}$-concept term C using only roles $r_1, \ldots, r_n$ can be translated into a term t_C over the signature Σ_{SLmO} by replacing each concept constant A by a corresponding free constants a, each concept variable X by a corresponding variable x, $\top$ by 1, $\sqcap$ by $\wedge$, and $\exists r_i$ by f_i. For example, the $\mathcal{EL}$-concept term $C = A \sqcap \exists r_1.\top \sqcap \exists r_3.(X \sqcap B)$ is translated into $t_C = a \wedge f_1(1) \wedge f_3(x \wedge b)$. Conversely, any term over the signature Σ_{SLmO} can be translated back into an $\mathcal{EL}$-concept term.

Lemma 5. *Let C, D be $\mathcal{EL}$-concept term using only roles $r_1, \ldots, r_n$. Then $C \equiv D$ iff $t_C =_{SLmO} t_D$.*

As an immediate consequence of this lemma, we have that unification in the DL $\mathcal{EL}$ corresponds to unification modulo the equational theory *SLmO*. Thus, Theorem 2 implies that *SLmO* has unification type zero, and Theorem 3 implies that *SLmO*-unification is NP-complete.

Corollary 2. *The equational theory SLmO of semilattices with monotone operators has unification type zero, and deciding solvability of an SLmO-unification problem is an NP-complete problem.*

7 Conclusion

In this paper, we have shown that unification in the DL $\mathcal{EL}$ is of type zero and NP-complete. There are interesting differences between the behaviour of $\mathcal{EL}$ and the closely related DL $\mathcal{FL}_0$ w.r.t. unification and matching. Though the unification types coincide for these two DLs, the complexities of the decision problems differ: $\mathcal{FL}_0$-unification is ExpTime-complete, and thus considerably harder than $\mathcal{EL}$-unification. In contrast, $\mathcal{FL}_0$-matching is polynomial, and thus considerably easier than $\mathcal{EL}$-matching, which is NP-complete.

It is well-known that there is a close connection between modal logics and DLs [5]. For example, the DL $\mathcal{ALC}$, which can be obtained by adding negation to $\mathcal{EL}$ or $\mathcal{FL}_0$, corresponds to the basic (multi-)modal logic K. Decidability of unification in K is a long-standing open problem. Recently, undecidability of unification in some extensions of K (for example, by the universal modality) was shown in [20]. The undecidability results in [20] also imply undecidability of unification in some expressive DLs (e.g., $\mathcal{SHIQ}$). The unification types of some modal (and related) logics have been determined by Ghilardi; for example in [14] he shows that $\mathsf{K4}$ and $\mathsf{S4}$ have unification type finitary. Unification in sub-Boolean modal logics (i.e., modal logics that are not closed under all Boolean operations, such as the modal logic equivalent of $\mathcal{EL}$) has, to the best of our knowledge, not been considered in the modal logic literature.

References

1. Baader, F.: Unification in commutative theories. J. of Symbolic Computation 8(5) (1989)
2. Baader, F.: Terminological cycles in KL-ONE-based knowledge representation languages. In: Proc. AAAI 1990 (1990)
3. Baader, F.: Terminological cycles in a description logic with existential restrictions. In: Proc. IJCAI 2003 (2003)
4. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. IJCAI 2005 (2005)
5. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
6. Baader, F., Küsters, R.: Matching in description logics with existential restrictions. In: Proc. KR 2000 (2000)
7. Baader, F., Küsters, R.: Unification in a description logic with transitive closure of roles. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250, p. 217. Springer, Heidelberg (2001)
8. Baader, F., Küsters, R., Borgida, A., McGuinness, D.L.: Matching in description logics. J. of Logic and Computation 9(3) (1999)
9. Baader, F., Narendran, P.: Unification of concepts terms in description logics. J. of Symbolic Computation 31(3) (2001)
10. Baader, F., Nutt, W.: Basic description logics. In: [5] (2003)
11. Baader, F., Sertkaya, B., Turhan, A.-Y.: Computing the least common subsumer w.r.t. a background terminology. J. of Applied Logic 5(3) (2007)
12. Baader, F., Snyder, W.: Unification theory. In: Handbook of Automated Reasoning, vol. I. Elsevier Science Publishers, Amsterdam (2001)
13. Brandt, S.: Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else. In: Proc. ECAI 2004 (2004)
14. Ghilardi, S.: Best solving modal equations. Ann. Pure Appl. Logic 102(3) (2000)
15. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics 1(1) (2003)
16. Kazakov, Y., de Nivelle, H.: Subsumption of concepts in $\mathcal{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is PSPACE-complete. In: Proc. DL 2003. CEUR Electronic Workshop Proceedings (2003), `http://CEUR-WS.org/Vol-81/`
17. Küsters, R.: Non-Standard Inferences in Description Logics. LNCS (LNAI), vol. 2100. Springer, Heidelberg (2001)
18. Rector, A., Horrocks, I.: Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In: Proc. AAAI 1997 (1997)
19. Sofronie-Stokkermans, V.: Locality and subsumption testing in $\mathcal{EL}$ and some of its extensions. In: Proc. AiML 2008 (2008)
20. Wolter, F., Zakharyaschev, M.: Undecidability of the unification and admissibility problems for modal and description logics. ACM Trans. Comput. Log. 9(4) (2008)

Unification with Singleton Tree Grammars

Adrià Gascón[1,⋆], Guillem Godoy[1,⋆], and Manfred Schmidt-Schauß[2]

[1] LSI Department, Universitat Politècnica de Catalunya Jordi Girona, 1-3 08034 Barcelona, Spain
adriagascon@gmail.com, ggodoy@lsi.upc.edu

[2] Dept. Informatik und Mathematik, Inst.f. Informatik, J.W. Goethe-University, D-60054 Frankfurt, Germany
schauss@ki.informatik.uni-frankfurt.de

Abstract. First-order term unification is an essential concept in areas like functional and logic programming, automated deduction, deductive databases, artificial intelligence, information retrieval, compiler design, etc. We build upon recent developments in general grammar-based compression mechanisms for terms, which are more general than dags and investigate algorithms for first-order unification of compressed terms.

We prove that the first-order unification of compressed terms is decidable in polynomial time, and also that a compressed representation of the most general unifier can be computed in polynomial time.

We use several known results on the used tree grammars, called singleton tree grammars (STG)s, like polynomial time computability of several subalgorithmms: certain grammar extensions, deciding equality of represented terms, and generating the preorder traversal. An innovation is a specialized depth of an STG that shows that unifiers can be represented in polynomial space.

1 Introduction

Solving equations is an important task in any mathematically founded science and deserves thorough investigations. In general, solving an equation $s \doteq t$ consists of finding a substitution σ for variables occurring in both expressions s and t such that $\sigma(s) = \sigma(t)$. The range of the variables, the kind of expressions s and t, and their semantics, as well as the semantics of $=$ depend on the context. By restricting some parameters we obtain the well-known first-order term unification problem, where the expressions s and t are terms with variables standing for terms, function symbols are uninterpreted, and $=$ is interpreted as syntactic equality. Therefore, the term unification problem asks for a substitution σ that maps the variables to *first-order* terms such that $\sigma(s)$ and $\sigma(t)$ are syntactically equal. For example, the first-order unification instance $f(f(x_2, x_2), f(x_3, x_3)) \doteq f(x_1, x_2)$ has a solution $\langle x_1 \mapsto f(f(x_3, x_3), f(x_3, x_3)), x_2 \mapsto f(x_3, x_3)\rangle$. First-order unification is efficiently solvable [MM82, BS01], and an essential algorithm

⋆ The first two authors were supported by Spanish Min. of Educ. and Science by the FORMALISM project (TIN2007-66523) and by the LOGICTOOLS-2 project (TIN2007-68093-C02-01).

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 365–379, 2009.
© Springer-Verlag Berlin Heidelberg 2009

in areas like functional and logic programming, automated deduction, deductive databases, artificial intelligence, information retrieval, compilers, etc.

However, many of the applications in the areas mentioned above deal with large data-objects. For this reason, some kind of internal succinct representation of terms is required in order to guarantee computability in an environment with a limited amount of resources. Therefore, it is important to reconsider complexity issues for the original problem and its variants applied to compressed input terms. In recent years there has been an increase of interest in compression mechanisms based on grammar representation, since other mechanisms can in general be efficiently simulated. These compression techniques were initially used for words [Pla95, Loh06, Lif07], and led to important results in string processing, with applications [HSTA00, GM02, LR06] in software/hardware verification, information retrieval, and bioinformatics. In that sense, *Straight-Line Programs (SLP)* or the equivalent formalism of *Singleton Context Free Grammars (SCFG)* are now a widely accepted formalism for text compression. Later, grammar-based compression was extended to terms/trees [BLM05, SS05, CDG$^+$97] with applications on XML tree structure compression [BLM05] and XPATH [LM05]. STG-based compressors have already been developed [MMS08]. Essentially, an SCFG, i.e. a context free grammar where all nonterminals generate a singleton language, is used for representing single words, and similarly, every nonterminal in a *singleton tree grammar (STG)* represents one tree. An STG can succinctly represent terms/trees which are exponentially big in size and height. Efficient algorithms have been developed for checking whether two compressed inputs represent the same word/term [Pla95, Loh06, Lif07], and for finding occurrences of one of them within the other (fully compressed pattern matching)[KRS95, KPR96, MST97, Lif07]. Recently, it was shown that tree grammars using multi-hole-contexts are polynomially equivalent to STGs [LMSS09]. STGs have also been used for complexity analysis of unification algorithms in [LSSV06b, LSSV06a], and the matching problem [GGSS08].

In this paper, we prove that first-order unification is decidable in polynomial time even when the input is compressed using STGs. Our algorithm generates the most general unifier in polynomial time and represents it again with an STG.

1.1 Outline of the Algorithm

The global structure of the algorithm is rather standard (see [Rob65]). Given two terms s and t, we look for a minimal position p where s and t differ. If s and t contain different function symbols at p, then we terminate stating that they are not unifiable. Otherwise, one of s or t, say s, contains a variable x at p. If x properly occurs in the subterm of t at p, then we terminate, again stating non-unifiability. Otherwise, we replace x by the subterm of t at p everywhere, and re-start the process again, until both s and t become equal, in which case we state unifiability.

The difficulties are induced by the task of performing all the operations mentioned above on the compressed representation of terms and positions. Positions are usually represented as sequences of integers, each one indicating the selected

child at each level from the root. Since STGs may represent terms with an exponential height, we also need to deal with compressed representations of positions in terms. In [LSSV06b, LSSV04, GGSS08], SCFGs are already used for this purpose.

We prove that all the needed operations can be performed in polynomial time with respect to the compressed representation of the terms with STG. Many of these operations can be done by an adequate use of previous results, since operations such as computing subterms, asking for the occurrence of a variable symbol in a certain subterm, or computing prefixes and suffixes of positions and contexts are known to be efficiently computable [GGSS08]. These basic operations on STGs and SCFGs are presented in Section 3.1. In [BLM05] it was shown how to succinctly represent the preorder traversal word of a word represented by an STG using an SCFG. Since we need to find a position where s and t differ, an option is to consider the first different symbol in s and t found while traversing them in preorder. To compute it, we represent the preorder traversals of s and t with words w_s and w_t compressed with an SCFG, find the first index i where w_s and w_t differ, and obtain p from i. We show how to perform all of these operations efficiently in Section 3.2.

We also need to apply substitutions once a variable is isolated. Performing a replacement of a first-order variable x by a term u is easily representable with STGs by simply transforming x into a non-terminal of the grammar and adding rules such that it generates u. However, since successive replacements of variables by subterms modify the initial terms, we have to show that this does not produce an exponential space increase of the grammar, since the depth of the grammar may be doubled after each of these operations. To this end, we develop a notion of restricted depth, showing that it is preserved along the execution, and that the size increase at each step can be bounded by this restricted depth, which is shown in Section 3.3. This improves upon the proof techniques used for showing polynomiality of the first-order matching algorithm on compressed terms.

2 Preliminaries

A *signature* is a set $\mathcal{F}$ along with a function $\mathtt{ar} : \mathcal{F} \to \mathbb{N}$. Members of $\mathcal{F}$ are called function symbols, and $\mathtt{ar}(f)$ is called the *arity* of the function symbol f. Function symbols of arity 0 are called constants. Let $\mathcal{X}$ be a set disjoint from $\mathcal{F}$ whose elements are called variables. The set $\mathcal{T}(\mathcal{F}, \mathcal{X})$ of terms over $\mathcal{F}$ and $\mathcal{X}$ is defined to be the smallest set containing $\mathcal{X}$ and having the property that $f(t_1, \ldots, t_m) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ whenever $f \in \mathcal{F}$, $m = \mathtt{ar}(f)$ and $t_1, \ldots, t_m \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

The size $|t|$ of a term t is the number of occurrences of variables and function symbols in t. The $\mathtt{height}$ of a term t is 0 if t is a constant or a variable, and $1 + \mathtt{max}\{\mathtt{height}(t_1), \ldots, \mathtt{height}(t_m)\}$ if $t = f(t_1, \ldots, t_m)$. The preorder traversal of a term t, denoted $\mathtt{pre}(t)$, is the word defined recursively by $\mathtt{pre}(f(t_1, \ldots, t_m)) = f \cdot \mathtt{pre}(t_1) \cdots \mathtt{pre}(t_m)$. *Positions* of a term t, denoted p, q, are sequences of natural numbers that are used to identify the position of subterms of t. The length of a position p is denoted by $|p|$. The set $Pos(t)$ of *positions* of t is defined by

$Pos(t) = \{\lambda\}$ if t is a constant or a variable, and $Pos(t) = \{\lambda\} \cup \{1 \cdot p \mid p \in Pos(t_1)\} \cup \ldots \cup \{m \cdot p \mid p \in Pos(t_m)\}$ if $t = f(t_1, \ldots, t_m)$, where λ denotes the empty sequence and $p \cdot q$, or simply pq, denotes the concatenation of p and q. If t is a term and p a position, then $t|_p$ is the subterm of t at position p. More formally defined, $t|_\lambda = t$ and $f(t_1, \ldots, t_m)|_{i \cdot p} = t_i|_p$. We denote by $t[s]_p$ the term that is like t except that the subterm $t|_p$ is replaced by s. More formally defined, $t[s]_\lambda = s$ and $f(t_1, \ldots, t_m)[s]_{i \cdot p} = f(t_1, \ldots, t_{i-1}, t_i[s]_p, t_{i+1}, \ldots, t_m)$. We can define a partial order $\leq$ on $Pos(t)$ by $p \leq q$ if and only if p is a prefix of q, i.e there is a sequence p' such that $q = p \cdot p'$. We say that positions p and q are *disjoint* if they are incomparable with respect to $\leq$. A *substitution* is a mapping $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$. Substitutions can also be applied to arbitrary terms by homomorphically extending them by $\sigma(f(t_1, \ldots, t_m)) = f(\sigma(t_1), \ldots, \sigma(t_m))$.

Intuitively, contexts are terms with a single occurrence of a hole $[\cdot]$ into which terms (or other contexts) may be inserted. We denote contexts by upper case letters C, D. We can provide a formal definition by considering a context to be a term in an extended signature that includes a single extra constant symbol $[\cdot]$. Then, if C and D are contexts and s is a term, CD and Cs represent the term that is like C except that the occurrence of $[\cdot]$ is replaced by D and s, respectively. If $D_1 = D_2 D_3$ for contexts D_i, then D_2 is called a *prefix* of D_1, and D_3 is called a *suffix* of D_1. The position of the hole in a context C is called *hole path*, and denoted $\mathtt{hp}(C)$, and its length is denoted as $|\mathtt{hp}(C)|$.

Definition 2.1. *A* singleton context-free grammar (SCFG) *G is a 3-tuple $\langle \mathcal{N}, \Sigma, R \rangle$, where $\mathcal{N}$ is a set of non-terminals, Σ is a set of symbols, and R is a set of rules of the form $N \to \alpha$ where $N \in \mathcal{N}$ and $\alpha \in (\mathcal{N} \cup \Sigma)^*$. The sets $\mathcal{N}$ and Σ must be disjoint, $|\{N \to \alpha \in R \mid \alpha \in (\mathcal{N} \cup \Sigma)^*\}| = 1$ for all N in $\mathcal{N}$, and the SCFG must be non-recursive, i.e., the transitive closure $>_G^+$ generated by all $N >_G M$ if $N \to \alpha_1 M \alpha_2 \in R$ must be terminating. The word generated by a non-terminal N of G, denoted by $w_{G,N}$ or w_N when G is clear from the context, is the word in Σ^* reached from N by successive applications of the rules of G.*

Usual definitions of SCFG require it to be in Chomsky normal form. We do not keep this restriction to ease the presentation. But note that our SCFG can be converted into the more standard ones by a linear transformation.

Definition 2.2. *A* singleton tree grammar (STG) *is a 4-tuple $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R)$, where $\mathcal{TN}$ is a set of tree/term non-terminals, or non-terminals of arity 0, $\mathcal{CN}$ is a set of context non-terminals, or non-terminals of arity 1, and Σ is a signature of function symbols (the terminals), such that the sets $\mathcal{TN}$, $\mathcal{CN}$, and Σ are pairwise disjoint. The set of non-terminals $\mathcal{N}$ is defined as $\mathcal{N} = \mathcal{TN} \cup \mathcal{CN}$. The rules in R may be of the form:*

- $A \to f(A_1, \ldots, A_m)$, *where* $A, A_i \in \mathcal{TN}$, *and* $f \in \Sigma$ *with* $\mathtt{ar}(f) = m$.
- $A \to C_1 A_2$ *where* $A, A_2 \in \mathcal{TN}$, *and* $C_1 \in \mathcal{CN}$.
- $C \to [\cdot]$ *where* $C \in \mathcal{CN}$.
- $C \to C_1 C_2$, *where* $C_i \in \mathcal{CN}$.

- $C \to f(A_1, \ldots, A_{i-1}, C_i, A_{i+1}, \ldots, A_m)$, *where* $A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_m \in \mathcal{TN}$, $C, C_i \in \mathcal{CN}$, *and* $f \in \Sigma$ *with* $\mathtt{ar}(f) = m$.
- $A \to A_1$, *(λ-rule) where A and A_1 are term non-terminals.*

Let $N_1 >_G N_2$ for two non-terminals N_1, N_2, iff $N_1 \to t$, and N_2 occurs in t. The STG must be non-recursive, i.e. the transitive closure $>_G^+$ must be terminating. Furthermore, for every non-terminal N there is exactly one rule having N as left-hand side. Given a term t with occurrences of non-terminals, the derivation of t by G is an exhaustive iterated replacement of the non-terminals by the corresponding right hand sides. The result is denoted as $w_{G,t}$. In the case of a non-terminal N we also say that N generates *$w_{G,N}$. We will write w_N when G is clear from the context.*

Note that we use Σ instead of $\mathcal{F}$ to also allow first-order variables as constants in Σ, which are not in $\mathcal{F}$. λ-rules are not necessary for compression. However, they will be useful when applying substitutions of variables to terms represented by STGs.

Definition 2.3. *The* size *$|G|$ of an STG (SCFG) G is the sum of the sizes of its rules, where the size of a rule $N \to \alpha$ is $1 + |\alpha|$. The* depth *within G of a non-terminal N is defined recursively as $\mathtt{depth}(N) := 1 + \mathtt{max}\{\mathtt{depth}(N') \mid N'$ is a non-terminal occurring in α where $N \to \alpha \in G\}$ and the empty maximum is assumed to be 0. The* depth of a grammar *is the maximum of the depths of all non-terminals, denoted as $\mathtt{depth}(G)$.*

Term dags can efficiently be represented in STGs by considering an empty set of context non-terminals, which corresponds to the commonly used implementation of dags by adjacency lists. However, STG-represented terms may have exponential depth in the size of the grammar in contrast to dags, which only allow for a linear depth in the (notational) size of the dags.

Example 2.4. Let $G = \{\{A, A_1\}, \{C', C_0, C_1, C_2, \ldots, C_{n-1}, C_n\}, \{f, a, x\}, R\}$, where $R = \{A \to C_n A_1, A_1 \to x, C_0 \to f(C'), C' \to [\cdot], C_1 \to C_0 C_0, C_2 \to C_1 C_1, \ldots, C_n \to C_{n-1} C_{n-1}\}$.
$\mathtt{depth}(G) = \mathtt{depth}(A) = n + 3$. The term non-terminal A of G represents the term $w_{G,A} = f^{2^n}(x)$, whose height and size are exponential with respect to $|G|$.

Plandowski [Pla95] proved decidability in polynomial time of the word problem for SCFG, i.e., given an SCFG P and two non-terminals A and B of P, to decide whether $w_{P,A} = w_{P,B}$. The best complexity for this problem has been recently obtained by Lifshits [Lif07] with time $\mathcal{O}(|P|^3)$. In [BLM05, SS05]) Plandowski's result is generalized to STG. Since the result in [BLM05] is based on a linear reduction from terms to words and a direct application of Plandowski's result, it also holds using the Lifshits result. Hence, we have the following.

Theorem 2.5. *([Lif07, BLM05]) Given an STG G, and two tree non-terminals A, B from G, it is decidable in time $\mathcal{O}(|G|^3)$ whether $w_A = w_B$.*

We will use more specific information from Lifshits' work [Lif07].

Lemma 2.6. *[Lif07] Let G be an SCFG. Then a data structure can be computed in time $\mathcal{O}(|G|^3)$ which allows to answer to the following question in time $\mathcal{O}(|G|)$: given two non-terminals N_1 and N_2 of G and an integer value k, does w_{N_1} occur in w_{N_2} at position k?*

The unification problem is a generalization of the word problem, since it allows occurrences of variables and substituting them in order to satisfy equality.

Definition 2.7. *The* first-order unification problem with STG *has an STG G representing first-order terms and contexts as input, plus two term non-terminals A_s and A_t of G representing terms $s = w_{G,A_s}$ and $t = w_{G,A_t}$. Its decisional version asks whether s and t are unifiable. In the affirmative case, its computational version asks for a representation of the most general unifier.*

Example 2.8. Let $G = (\{A_t, A_s, A, B, A', B'\}, \{C_0, C_1, C_2, C_3, C_4, C', D\}, \{g, f, a, x\}, R)$, where $R = \{A_t \to g(B, A), A_s \to g(A, A), A \to C_4[A'], C_4 \to C_3C_3, C_3 \to C_2C_2, C_2 \to C_1C_1, C_1 \to C_0C_0, C_0 \to f(C'), C' \to [\cdot], A' \to a\ B \to D[B'], D \to C_3C_2, B' \to x\}$, be an STG. Note that $w_{G,A_t} = g(f^{12}(x), f^{16}(a))$, and $w_{G,A_s} = g(f^{16}(a), f^{16}(a))$. Hence, $\langle G, A_s, A_t\rangle$ is an instance of first-order unification with STG. The goal is to find a substitution σ such that $\sigma(w_{G,A_s}) = \sigma(w_{G,A_t})$.

3 Basic Operations with STG and SCFG

Usual term unification algorithms need to compute subterms, apply substitutions, look for the difference between two terms, look for the occurrences of a certain variable, etc. We need to perform these operations when the input terms are represented by an STG. We use SCFGs for concisely representing positions of a term represented with an STG. For clarity we call the non-terminals of this SCFG *position non-terminals*.

3.1 Known Results

We give a list of operations which are known to be computable in linear time for a given STG G generating terms, and an SCFG P generating positions.

- For every position non-terminal p of P and non-terminal N of G, the numbers $|w_p|$ and $|w_N|$ are computable in time $\mathcal{O}(|P|)$ and $\mathcal{O}(|G|)$, respectively.
- An SCFG H_G can be computed in time $\mathcal{O}(|G|)$ for G such that, for every context non-terminal $C \in G$, there exists a position non-terminal H_C of H_G which generates $\mathtt{hp}(w_C)$. Moreover, the depth of every H_C is the same as the one of its corresponding C, and $\mathtt{depth}(H_G) \leq \mathtt{depth}(G)$.
- Given a position non-terminal p of P and a number $l \leq |w_p|$, the SCFG P can be extended in time $\mathcal{O}(|P|)$ with $\mathtt{depth}(p)$ new non-terminals such that one of them, called p' generates the prefix (suffix) of w_p of length l. Moreover, $\mathtt{depth}(p') \leq \mathtt{depth}(p)$, and the new SCFG P' satisfies $\mathtt{depth}(P') = \mathtt{depth}(P)$.

We present in detail the case of the extension of G generating a certain suffix of w_C, for a context non-terminal C of G given in [GGSS08], as a definition here.

Definition 3.1. *Let G be an STG and let C be a context non-terminal of G. Let l be a natural number such that $l \leq |\mathtt{hp}(w_C)|$. Then, we define $\mathtt{Suff}(G, C, l)$ as an extension of G recursively as follows.*

- *If $l = 0$, then $\mathtt{Suff}(G, C, l) := G$. In the next cases we assume $l > 0$.*
- *If $(C \to C_1 C_2) \in G$ and $l < |\mathtt{hp}(w_{C_1})|$. Then $\mathtt{Suff}(G, C, l)$ includes $\mathtt{Suff}(G, C_1, l)$, which contains a non-terminal C_1' generating the suffix of w_{C_1} with $|\mathtt{hp}(w_{C_1'})| = |\mathtt{hp}(w_{C_1})| - l$, plus the rule $C' \to C_1' C_2$, where C' is an additional new non-terminal.*
- *If $(C \to C_1 C_2) \in G$ and $l \geq |\mathtt{hp}(w_{C_1})|$, then, with $l' := l - |\mathtt{hp}(w_{C_1})|$, we define $\mathtt{Suff}(G, C, l)$ as $\mathtt{Suff}(G, C_2, l')$.*
- *If $(C \to f(A_1, \ldots, A_{i-1}, C_i, A_{i+1}, \ldots, A_m)) \in G$, then we define $\mathtt{Suff}(G, C, l)$ as $\mathtt{Suff}(G, C_i, l - 1)$.*
- *In any other case $\mathtt{Suff}(G, C, l)$ is undefined.*

Lemma 3.2. *Let G be an STG describing first-order terms and contexts. Let C be a context non-terminal of G, and let l be a natural number such that $l \leq |\mathtt{hp}(w_C)|$.*

Then, $G' = \mathtt{Suff}(G, C, l)$ is computable in time $\mathcal{O}(|G|)$, it adds at most $\mathtt{depth}(C)$ new context non-terminals, and one context non-terminal C' of G' generates the suffix of w_C with $|\mathtt{hp}(w_{C'})| = |\mathtt{hp}(w_C)| - l$. Moreover, for every new non-terminal N, $\mathtt{depth}_{G'}(N) \leq \mathtt{depth}_G(C)$, and $\mathtt{depth}(G') = \mathtt{depth}(G)$.

Extending G to generate $w_N|_{w_p}$, for a non-terminal N of G and a non-terminal position p of P, is known to be computable in polynomial time. This was shown in [GGSS08]. We present such an extension as a definition here.

Definition 3.3. *Let G be an STG describing first-order terms and contexts, and let P be an SCFG describing positions. Let p be a position non-terminal of P and N a non-terminal of G. We recursively define $\mathtt{pExt}(G, N, p, P)$ as an extension of G as follows.*

- *If $w_p = \lambda$ (the empty word), then $\mathtt{pExt}(G, N, p, P) := G$. In the next cases we assume $w_p \neq \lambda$.*
- *If $(N \to C_1 N_2) \in G$ and $w_p < \mathtt{hp}(w_{C_1})$, then $\mathtt{pExt}(G, N, p, P)$ includes $\mathtt{Suff}(G, C_1, |w_p|)$, which contains a non-terminal C_1' generating the suffix of w_{C_1} with $|\mathtt{hp}(w_{C_1'})| = |\mathtt{hp}(w_{C_1})| - |w_p|$, plus the rule $N' \to C_1' N_2$, where N' is an additional new non-terminal.*
- *If $(N \to C_1 N_2) \in G$ and w_p is disjoint from $\mathtt{hp}(w_{C_1})$, then $\mathtt{pExt}(G, N, p, P) := \mathtt{pExt}(G, C_1, p, P)$.*
- *If $(N \to C_1 N_2) \in G$ and $\mathtt{hp}(w_{C_1}) \leq w_p$, then extend P with $\mathtt{depth}(p)$ new non-terminals where one of them called p' generates the suffix of w_p of length $|w_p| - |\mathtt{hp}(w_{C_1})|$, and define $\mathtt{pExt}(G, N, p, P) := \mathtt{pExt}(G, N_2, p', P)$.*

- *If* $(N \to f(N_1, \ldots, N_m)) \in G$, *and* $i \leq w_p$ *with* $1 \leq i \leq m$, *then extend* P *with* $\texttt{depth}(p)$ *new non-terminals where one of them called* p' *generates the suffix of* w_p *of length* $|p| - 1$, *and define* $\texttt{pExt}(G, N, p, P) := \texttt{pExt}(G, N_i, p', P)$.
- *In any other case* $\texttt{pExt}(G, N, p, P)$ *is undefined.*

Lemma 3.4. *Let* G *be an STG describing first-order terms and contexts, and* P *be an SCFG describing positions. Let* p *be a position non-terminal of* P, *and* N *a non-terminal of* G.

It can be checked in time $\mathcal{O}((|G| + |P|)^4)$ *if* w_p *is a valid position of* w_N.

Moreover, $G' = \texttt{pExt}(G, N, p, P)$ *is computable in time* $\mathcal{O}((|G| + |P|)^4)$, *it adds at most* $\texttt{depth}(N)$ *new non-terminals, and one non-terminal of* G' *generates* $w_N|_{w_p}$. *Furthermore, for every new non-terminal* N', $\texttt{depth}_{G'}(N') \leq \texttt{depth}_G(N)$, *and* $\texttt{depth}(G') = \texttt{depth}(G)$.

3.2 Finding the First Different Position of Two Terms

Given two terms s and t, represented by term non-terminals A_s and A_t of an STG G, we show how to efficiently construct a succinct SCFG P with a position non-terminal p generating the word w_p, which represents the first different position of s and t, i.e. the first one with a different root symbol found when we traverse them in preorder. Recall that such a word is a position in s and t, thus a sequence of integers. The SCFG P is obtained in three steps detailed in the next three subsections. We first construct an SCFG $\texttt{Pre}_G$ with non-terminals $\mathcal{P}_s$ and $\mathcal{P}_t$ generating the preorder traversals $\texttt{pre}(s)$ and $\texttt{pre}(t)$ of s and t, respectively. This is based on the ideas of [BLM05]. Then, given $\mathcal{P}_s$ and $\mathcal{P}_t$, we describe a procedure to efficiently compute the first index k in which $\texttt{pre}(s)$ and $\texttt{pre}(t)$ differ. Finally, given the index k we show how to construct the desired SCFG P.

Computing the preorder traversal of a term. Two arbitrary different trees may have the same preorder traversal, but when they represent terms over a fixed signature where the arity of every function symbol is fixed, the preorder traversal is unique for every term. Given a term t, there is a natural bijective mapping between the indexes $\{1, \ldots, |\texttt{pre}(t)|\}$ of $\texttt{pre}(t)$ and the positions $\texttt{Pos}(t)$ of t, which associates every position $p \in \texttt{Pos}(t)$ to the index $i \in \{1, \ldots, |\texttt{pre}(t)|\}$ you find at $\texttt{root}(t|p)$ while traversing the tree in preorder. We can recursively define the two mappings $\texttt{pIndex}(t, p) \to \{1, \ldots, |\texttt{pre}(t)|\}$ and $\texttt{iPos}(t, i) \to \texttt{Pos}(t)$ as follows. $\texttt{pIndex}(t, \lambda) = 1$, $\texttt{pIndex}(f(t_1, \ldots, t_m), i.p) = (1 + |t_1| + \ldots + |t_{i-1}|) + \texttt{pIndex}(t_i, p)$, $\texttt{iPos}(t, 1) = \lambda$, and $\texttt{iPos}(f(t_1, \ldots, t_m), 1 + |t_1| + \ldots + |t_{i-1}| + k) = i.\texttt{iPos}(t_i, k)$ for $1 \leq k \leq |t_i|$.

In [BLM05] it is shown how to construct, from a given STG G, an SCFG $\texttt{Pre}_G$ representing the preorder traversals of the terms generated by G. We reproduce that construction here, presented in Figure 1 as a set of rules indicating, for each term non-terminal A and its rule $A \to \alpha$ of G, which rule $\mathcal{P}_A \to \alpha'$ of $\texttt{Pre}_G$ is required in order to make the non-terminal $\mathcal{P}_A$ of $\texttt{Pre}_G$ satisfy $w_{\texttt{Pre}_G, \mathcal{P}_A} =$

$$
\begin{aligned}
A \to f(A_1, \ldots, A_m) &\Rightarrow \mathcal{P}_A \to f\mathcal{P}_{A_1} \ldots \mathcal{P}_{A_m} \\
A \to C_1 A_2 &\Rightarrow \mathcal{P}_A \to \mathcal{L}_{C_1} \mathcal{P}_{A_2} \mathcal{R}_{C_1} \\
A \to A_1 &\Rightarrow \mathcal{P}_A \to \mathcal{P}_{A_1} \\
C \to C_1 C_2 &\Rightarrow \begin{cases} \mathcal{L}_C \to \mathcal{L}_{C_1} \mathcal{L}_{C_2} \\ \mathcal{R}_C \to \mathcal{R}_{C_2} \mathcal{R}_{C_1} \end{cases} \\
C \to f(A_1, \ldots, A_{i-1}, C_i, A_{i+1}, \ldots, A_m) &\Rightarrow \begin{cases} \mathcal{L}_C \to f\mathcal{P}_{A_1} \ldots \mathcal{P}_{A_{i-1}} \mathcal{L}_{C_i} \\ \mathcal{R}_C \to \mathcal{R}_{C_i} \mathcal{P}_{A_{i+1}} \ldots \mathcal{P}_{A_n} \end{cases} \\
C \to [\cdot] &\Rightarrow \begin{cases} \mathcal{L}_C \to \lambda \\ \mathcal{R}_C \to \lambda \end{cases}
\end{aligned}
$$

Fig. 1. Generating the Preorder Traversal

$$
\texttt{index}(p_1,p_2,k',P) = \begin{cases} 1 & \text{, if } |w_{p_1}| = 1 \\ \texttt{index}(p_{11},p_2,k',P) & \text{, if } (p_1 \to p_{11}p_{12}) \in P \wedge \\ & w_{p_{11}} \neq w_{p_2}[(k'+1)\ldots(k'+|w_{p_{11}}|)] \\ |w_{p_{11}}| + & \text{, if } (p_1 \to p_{11}p_{12}) \in P \wedge \\ \texttt{index}(p_{12},p_2,k'+|w_{p_{11}}|,P) & w_{p_{11}} = w_{p_2}[(k'+1)\ldots(k'+|w_{p_{11}}|)] \end{cases}
$$

Fig. 2. Algorithm for the Index of the First Difference

$\texttt{pre}(w_{G,A})$. To this end, for each context non-terminal C of G we also need non-terminals of $\texttt{Pre}_G$ generating the preorder traversal to the left of the hole ($\mathcal{L}_C$), and the preorder traversal to the right of the hole ($\mathcal{R}_C$).

It is straightforward to verify by induction on the depth of G that, for every term non-terminal A of G, the corresponding newly generated non-terminal $\mathcal{P}_N$ of $\texttt{Pre}_G$ generates $\texttt{pre}(w_N)$.

Example 3.5. (Continuation of Example 2.8) The SCFG $\texttt{Pre}_G$ obtained by applying the rules of Figure 1 to the STG G of Example 2.8 is $\{\mathcal{P}_{A_t} \to g\mathcal{P}_B\mathcal{P}_A, \mathcal{P}_{A_s} \to g\mathcal{P}_A\mathcal{P}_A, \mathcal{P}_A \to \mathcal{L}_{C_4}\mathcal{P}_{A'}\mathcal{R}_{C_4}, \mathcal{P}_{A'} \to a, \mathcal{P}_B \to \mathcal{L}_D\mathcal{P}_{B'}\mathcal{R}_D, \mathcal{L}_D \to \mathcal{L}_{C_3}\mathcal{L}_{C_2}, \mathcal{R}_D \to \mathcal{R}_{C_2}\mathcal{R}_{C_3}, \mathcal{P}_{B'} \to x, \mathcal{L}_{C_4} \to \mathcal{L}_{C_3}\mathcal{L}_{C_3}, \mathcal{L}_{C_3} \to \mathcal{L}_{C_2}\mathcal{L}_{C_2}, \mathcal{L}_{C_2} \to \mathcal{L}_{C_1}\mathcal{L}_{C_1}, \mathcal{L}_{C_1} \to \mathcal{L}_{C_0}\mathcal{L}_{C_0}, \mathcal{L}_{C_0} \to f\mathcal{L}_{C'}, \mathcal{L}_{C'} \to \lambda, \mathcal{R}_{C'} \to \lambda, \mathcal{R}_{C_0} \to \mathcal{R}_{C'}, \mathcal{R}_{C_1} \to \mathcal{R}_{C_0}\mathcal{R}_{C_0}, \mathcal{R}_{C_2} \to \mathcal{R}_{C_1}\mathcal{R}_{C_1}, \mathcal{R}_{C_3} \to \mathcal{R}_{C_2}\mathcal{R}_{C_2}, \mathcal{R}_{C_4} \to \mathcal{R}_{C_3}\mathcal{R}_{C_3}\}$. Note that $w_{\mathcal{P}_{A_t}} = gf^{12}xf^{16}a$ and $w_{\mathcal{P}_{A_s}} = gf^{16}af^{16}a$.

Computing the first different position of two words. Given two non-terminals p_1 and p_2 of an SCFG P, we want to find the first position k where w_{p_1} and w_{p_2} are different. In order to solve this problem, a linear search over the generated words is not a good idea, since their sizes may be exponentially big with respect to the size of P. But we can take advantage from Lemma 2.6 to make it faster. Thus, assume that the pre-computation of Lemma 2.6 has been done (in time $\mathcal{O}(|P|^3)$), and hence we can answer whether a given w_{p_1} occurs in a given w_{p_2} at a certain position in time $\mathcal{O}(|P|)$.

For finding the first different position between p_1 and p_2, we can assume $|w_{p_1}| \leq |w_{p_2}|$ without loss of generality. Moreover, we also assume $w_{p_1} \neq w_{p_2}[1 \ldots |w_{p_1}|]$ (with $w[i]$ we denote the symbol occurring at position i in the

$$\frac{(N \to \alpha) \wedge (k = 1)}{\mathcal{P}_{N,k} \to \lambda}$$

$$\frac{(N \to f(N_1, \ldots, N_m)) \wedge (1 + |w_{N_1}| + \ldots + |w_{N_{i-1}}| = k' < k \leq k' + |w_{N_i}|)}{\mathcal{P}_{N,k} \to i\mathcal{P}_{N_i,k-k'}}$$

$$\frac{(N \to C_1 N_2) \wedge (1 < k \leq |w_{\mathcal{L}_{C_1}}|)}{\mathcal{P}_{N,k} \to \mathcal{P}_{C_1,k}}$$

$$\frac{(N \to C_1 N_2) \wedge (k' = |w_{\mathcal{L}_{C_1}}| < k \leq |w_{\mathcal{L}_{C_1}}| + |w_{N_2}|)}{\mathcal{P}_{N,k} \to \mathcal{H}_{C_1}\mathcal{P}_{N_2,k-k'}}$$

$$\frac{(N \to C_1 N_2) \wedge (|w_{\mathcal{L}_{C_1}}| + |w_{N_2}| < k) \wedge (k' = |w_{N_2}|)}{\mathcal{P}_{N,k} \to \mathcal{P}_{C_1,k-k'+1}}$$

Fig. 3. Construction of the SCFG generating the position corresponding to the k-th index in $\texttt{pre}(w_N)$

word w, and with $w[i \ldots j]$ we denote the subword of w at position i and length $j - i + 1$). Note that this condition is necessary for the existence of a different position between w_{p_1} and w_{p_2}, and that this will be the case when p_1 and p_2 generate the preorder traversals of different trees. Finally, we can assume that P is in Chomsky Normal Form. Note that, if this was not the case, we can force this assumption with a linear time and space transformation.

We generalize our problem to the following question: given two non-terminals p_1 and p_2 of P and an integer k' satisfying $k' + |w_{p_1}| \leq |w_{p_2}|$ and $w_{p_1} \neq w_{p_2}[(k' + 1) \ldots (k' + |w_{p_1}|)]$, which is the smallest $k \geq 1$ such that $w_{p_1}[k]$ is different from $w_{p_2}[k' + k]$? (Note that we recover the original question by fixing $k' = 0$).

This generalization is solved efficiently by the recursive algorithm given in Figure 2, as can be shown inductively on the depth of p_1. By Lemma 2.6, each call takes time $\mathcal{O}(|P|)$, and at most $\texttt{depth}(P)$ calls are executed. Thus, the most expensive part of computing the first different position of w_{p_1} and w_{p_2} is the pre-computation given by Lemma 2.6, that is, $\mathcal{O}(|P|^3)$.

Lemma 3.6. *Let P be an SCFG of size n, and let p_1, p_2 be non-terminals of P such that $w_{p_1} \neq w_{p_2}$. The first position k where w_{p_1} and w_{p_2} differ is computable in time $\mathcal{O}(|P|^3)$.*

Example 3.7. (Continuation of Example 3.5) The SCFG $\texttt{Pre}_G$ is not in Chomsky Normal Form, but it is easy to adapt the algorithm of Figure 2 to this case. Thus, if we execute an adapted version of $\texttt{index}(\mathcal{P}_{A_t}, \mathcal{P}_{A_s}, 0, \texttt{Pre}_G)$, the following sequence of calls is produced: $\texttt{index}(\mathcal{P}_{A_t}, \mathcal{P}_{A_s}, 0, \texttt{Pre}_G)$, $\texttt{index}(\mathcal{P}_B, \mathcal{P}_{A_s}, 1, \texttt{Pre}_G)$, $\texttt{index}(\mathcal{P}_{B'}, \mathcal{P}_{A_s}, 13, \texttt{Pre}_G)$. The third call returns 1, the second one returns 13, and the first one returns 14, which corresponds to the first different position of w_{A_s} and w_{A_t}.

Computing the first different position of two terms. Using the index k from the previous subsection, we want to compute the SCFG P with a non-terminal p generating the word w_p, which represents the first different position

of s and t. Generalizing this, for a given non-terminal N of G and a given positive natural number k, we want to construct an SCFG with a non-terminal $\mathcal{P}_{N,k}$ generating the position of the symbol corresponding to the k-th index of $\texttt{pre}(w_N)$. In particular, $\mathcal{P}_{A_t,k}$ is the p we are looking for. We show how to do that again with a set of inference rules in Figure 3. These inference rules have to be understood to act on demand, i.e. they are executed to generate new non-terminals if those non-terminals have been demanded by other inference rules, or correspond to the initially demanded $\mathcal{P}_{A_t,k}$. Note that we make use here of the SCFG H_G, which can be constructed in linear time as commented in Section 3.1.

It is again straightforward to check, by induction on $\texttt{depth}(N)$, that every $\mathcal{P}_{N,k}$ generates $\texttt{iPos}(w_N, k)$. Note that the grammar rule of every of such $\mathcal{P}_{N,k}$ demands the existence of a grammar rule for at most one new $\mathcal{P}_{N',k'}$, and the corresponding N' satisfies $\texttt{depth}(N') < \texttt{depth}(N)$. Therefore, at most $\texttt{depth}(G)$ of the inference rules are executed for constructing $\mathcal{P}_{A_t,k}$.

The following lemma is a consequence of the three previous subsections.

Lemma 3.8. *Let G be an STG, and let A_s and A_t be term non-terminals of G such that $w_{A_s} \neq w_{A_t}$. Then, an SCFG P with a position non-terminal p generating the first different position in w_{A_s} and w_{A_t} can be computed in time $\mathcal{O}(|G|^3)$. Moreover, $|P| \leq |G|$ and $\texttt{depth}(P) \leq \texttt{depth}(G)$.*

Example 3.9. (Continuation of Example 3.7) We compute now an SCFG P with a position non-terminal $\mathcal{P}_{A_s,14}$ generating the position in w_{A_s} that corresponds to the 14^{th} index in its preorder traversal. We use the fact that the SCFG H_G presented in Section 3.1, can be constructed in linear time. The set of rules of H_G is $\{\mathcal{H}_{C_4} \to \mathcal{H}_{C_3}\mathcal{H}_{C_3}, \mathcal{H}_{C_3} \to \mathcal{H}_{C_2}\mathcal{H}_{C_2}, \mathcal{H}_{C_2} \to \mathcal{H}_{C_1}\mathcal{H}_{C_1}, \mathcal{H}_{C_1} \to \mathcal{H}_{C_0}\mathcal{H}_{C_0}, \mathcal{H}_{C_0} \to 1\mathcal{H}_{C'}, \mathcal{H}_{C'} \to \lambda, \mathcal{H}_D \to \mathcal{H}_{C_3}\mathcal{H}_{C_2}\}$. We construct P using the inference system presented in this section with the STG G, $N = A_s$, and $k = 14$. The set of rules of P is $\{\mathcal{P}_{A_s,14} \to 1\mathcal{P}_{A,13}, \mathcal{P}_{A,13} \to \mathcal{P}_{C_4,13}, \mathcal{P}_{C_4,13} \to \mathcal{H}_{C_3}\mathcal{P}_{C_3,5}, \mathcal{P}_{C_3,5} \to \mathcal{H}_{C_2}\mathcal{P}_{C_2,1}, \mathcal{P}_{C_2,1} \to \lambda\}$. Note that $w_{\mathcal{P}_{A_s,14}} = 11^8 1^4 \lambda = 1^{13}$.

3.3 Application of Substitutions and a Notion of Restricted Depth

Term unification algorithms usually apply substitutions when one variable is isolated. We need to emulate such applications when the terms are represented with STGs. In an STG, first-order variables are terminals of arity 0. Replacing a first-order variable X can be emulated by transforming X into a term non-terminal and adding the necessary rules for making X generate the replaced value. We define this notion of application of a substitution as follows.

Definition 3.10. *Let G be an STG. Let X be a terminal representing a first-order variable and let A be a term non-terminal of G, respectively. Then, $\{X \mapsto A\}(G)$ is defined as the STG obtained by adding the rule $X \to A$ to G, and converting X into a term non-terminal.*

Example 3.11. (Continuation of Example 3.9) We now apply three operations to the STG G given as input. We first extend G using the `pExt` construction

presented in Definition 3.3 such that a new non-terminal, called A'_s, generates $w_{A_s}|_{w_{\mathcal{P}_{A_s},14}}$. Then, we need to check that the variable x does not occur in $w_{A'_s}$, which can be done in linear time.Finally, we perform the substitution $\{x \mapsto A'_s\}(G)$ as stated in Definition 3.10. The set of rules of the obtained grammar G' after the `pExt` construction and this assignment is $\{\mathbf{x} \to \mathbf{A'_s}, \mathbf{A'_s} \to \mathbf{C_2 A'}, A_t \to g(B, A), A_s \to g(A, A), A \to C_4[A'], C_4 \to C_3C_3, C_3 \to C_2C_2, C_2 \to C_1C_1, C_1 \to C_0C_0, C_0 \to f(C'), C' \to [\cdot], A' \to a\ B \to D[B'], D \to C_3C_2, B' \to x\}$. The rules marked with bold correspond to the added non-terminals with respect to the initial STG G. Note that $w_{G',A'_s} = w_{G,A_s}|_{w_{P,\mathcal{P}_{A_s},14}} = w_{G,A_s}|_{1^{13}} = f^4(a)$, and thus, $w_{G',A_t} = g(f^{12}(w_{G',x}), f^{16}(a)) = g(f^{12}(w_{G',A'_s}), f^{16}(a)) = g(f^{12}f^4(a), f^{16}(a)) = g(f^{16}(a), f^{16}(a)) = w_{G',A_s}$. Hence, we state unifiability. The solution σ is represented in the STG G'.

When one or more substitutions of this form are applied, in general the depth of the non-terminals of G might increase. In order to see that the size increase is polynomially bounded along several substitution operations when unifying, we need a new notion of depth called `Vdepth`, which does not increase after an application of a substitution. It allows us to bound the final size increase of G. The notion of `Vdepth` is similar to the notion of `depth`, but it is 0 for the non-terminals N belonging to a special set V satisfying the following condition.

Definition 3.12. *Let $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R)$ be an STG, and let V be a subset of $\mathcal{TN} \cup \Sigma$. We say that V is a λ-*set *for G if for each term non-terminal A in V, the rule of G of the form $A \to \alpha$ is a λ-rule.*

Definition 3.13. *Let $G = (\mathcal{TN}, \mathcal{CN}, \Sigma, R)$ be an STG and let V be a λ-set for G. For every non-terminal N of G, the value $\texttt{Vdepth}_{G,V}(N)$, denoted also as $\texttt{Vdepth}_V(N)$ or $\texttt{Vdepth}(N)$ when G and/or V are clear from the context, is defined as follows (recall the convention that $\max(\emptyset) = 0$).*

$$\texttt{Vdepth}(N) := 0 \text{ for } N \in V$$
$$\texttt{Vdepth}(N) := 1 + \mathit{max}\{\texttt{Vdepth}(N') \mid N' \text{ is a non-terminal occurring in } \alpha, \text{ where } N \to \alpha \in G\}, \text{ otherwise.}$$

The `Vdepth` *of G is the maximum of the* `Vdepth` *of its non-terminals.*

The idea is to make V to contain all first-order variables, before and after converting them into term non-terminals. The following lemma is completely straightforward from the above definitions, and states that a substitution application does not modify the `Vdepth` provided $X \in V$ for the substitution $X \mapsto A$.

Lemma 3.14. *Let G, V be as in the above definition. Let $X \in V$ be a terminal of G of arity 0, and let A be a term non-terminal of G. Let G' be $\{X \mapsto A\}(G)$. Then, for any non-terminal N of G it holds that $\texttt{Vdepth}_{G'}(N) = \texttt{Vdepth}_G(N)$.*

We also need the fact that `Vdepth` does not increase due to the construction of $\texttt{pExt}(G, A, p, P)$ from G. This is stated by the following two lemmas.

```
Input: An STG G and term non-terminals A_s and A_t.
         (we write s and t for w_{A_s} and w_{A_t}).
While s and t are different do:
   Look for the first position p such that root(s|_p) ≠ root(t|_p).
   If both root(s|_p) and root(t|_p) are function symbols; Then
      Halt stating that the initial s and t are not unifiable
   Here, either s|_p or t|_p is a variable x, say s|_p, and both are different.
   If x occurs in t|_p Then
      Halt stating that the initial s and t are not unifiable
   Extend the compressed representation by the assignment {x ↦ t|_p}
EndWhile
Halt stating that the initial s and t are unifiable
```

Fig. 4. Unification Algorithm of STG-Compressed Terms

Lemma 3.15. *Let G be an STG, let C be a context non-terminal of G, let V be a set of terminals and term non-terminals of G, let l be a natural number smaller than $|\mathtt{hp}(w_C)|$, and let G' be $\mathtt{Suff}(G, C, l)$.*

Then, for every non-terminal N of G it holds that $\mathtt{Vdepth}_G(N) = \mathtt{Vdepth}_{G'}(N)$, and for every new non-terminal N in G' and not in G, it holds that $\mathtt{Vdepth}_{G'}(N) \leq \mathtt{Vdepth}_G(C)$. Moreover, the number of new added non-terminals is bounded by $\mathtt{Vdepth}_G(N)$.

Lemma 3.16. *Let G be an STG and P an SCFG, let N be a non-terminal of G, let V be a λ-set for G, let p be a position non-terminal of P such that $w_p \in \mathtt{Pos}(w_N)$, and let G' be $\mathtt{pExt}(G, N, p, P)$.*

Then, for every non-terminal N' of G it holds that $\mathtt{Vdepth}_G(N') = \mathtt{Vdepth}_{G'}(N')$, and for every new non-terminal N'' in G' and not in G, it holds that $\mathtt{Vdepth}_{G'}(N'') \leq \mathtt{Vdepth}_G(N)$. Moreover, the number of new added non-terminals is bounded by $\mathtt{Vdepth}_G(N)$.

4 A Polynomial Time Algorithm for First-Order Unification with STG

From a high level perspective the structure of our algorithm given in Figure 4 is very simple and rather standard. Most algorithms for first-order unification are variants of the above scheme. They represent the terms with directed acyclic graphs (dags), implemented somehow, in order to avoid the space explosion due to the repeated instantiation of variables by terms. In our setting, those terms are represented by STGs. In fact, the input is an STG G, and two term non-terminals A_s and A_t representing s and t, respectively. In the previous section we have already seen how to perform the basic required operations on STGs: look for the first position p satisfying that $\mathtt{root}(s|_p)$ and $\mathtt{root}(t|_p)$ are different, construct the term $t|_p$, and replace the variable $x = s|_p$ by $t|_p$ everywhere.

The algorithm runs in polynomial time due to the following observations. Let n and m be the initial value of $\mathtt{depth}(G)$ and $|G|$, respectively. We define

V to be the set of all the first-order variables at the start of the execution (before any of them has been converted into a non-terminal). Hence, at this point $\texttt{Vdepth}(G) = n$. The value $\texttt{Vdepth}(G)$ is preserved to be n along the execution of the algorithm thanks to Lemmas 3.14 and 3.16. Moreover, by Lemma 3.16, at most n new non-terminals are added at each step. Since at most $|V|$ steps are executed, the final size of G is bounded by $m + |V|n$. Each execution step takes time at most $\mathcal{O}(|G|^4)$. Thus we have proved:

Theorem 4.1. *First-order unification of two terms represented by an STG can be done in polynomial time ($\mathcal{O}(|V|(m + |V|n)^4)$, where m represents the size of the input STG, n represents the depth, and V represents the set of different first-order variables occurring in the input terms). This holds for the decision question, as well as for the computation of the most general unifier, whose components are represented by the final STG.*

5 Conclusion and Further Research

We presented an instantiation-based first-order unification algorithm, that can be immediately executed on the compressed representation of large terms and runs in polynomial time on the size of the representation.

Further research is to investigate extensions of first-order unification on compressed terms, and to investigate optimizations. Perhaps it is possible to show an improved upper bound. We believe that our techniques could be useful to decide the one context unification problem in NP when the input is represented by an STG. This problem has been solved for plain terms as input in [GGSST08].

Acknowledgement. We thank Markus Lohrey for valuable discussions on the subject of this paper.

References

[BLM05] Busatto, G., Lohrey, M., Maneth, S.: Efficient memory representation of XML documents. In: Bierman, G., Koch, C. (eds.) DBPL 2005. LNCS, vol. 3774, pp. 199–216. Springer, Heidelberg (2005)

[BS01] Baader, F., Snyder, W.: Unification theory. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 445–532. Elsevier/ MIT Press (2001)

[CDG+97] Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (1997), `http://www.grappa.univ-lille3.fr/tata` (release 1.10.2002)

[GGSS08] Gascón, A., Godoy, G., Schmidt-Schauß, M.: Context matching for compressed terms. In: 23rd IEEE LICS, pp. 93–102 (2008), `http://www.lsi.upc.edu/~ggodoy/publications.html`

[GGSST08] Gascón, A., Godoy, G., Schmidt-Schauß, M., Tiwari, A.: Context Unification with One Context Variable (to be published) (2008)

[GM02] Genest, B., Muscholl, A.: Pattern matching and membership for hierarchical message sequence charts. In: Rajsbaum, S. (ed.) LATIN 2002. LNCS, vol. 2286, pp. 326–340. Springer, Heidelberg (2002)

[HSTA00] Hirao, M., Shinohara, A., Takeda, M., Arikawa, S.: Fully compressed pattern matching algorithm for balanced straight-line programs. In: SPIRE 2000, p. 132. IEEE Computer Society Press, Washington (2000)

[KPR96] Karpinski, M., Plandowski, W., Rytter, W.: Efficient algorithms for Lempel-Ziv encoding. In: Proc. 4th Scandinavian Workshop on Algorithm Theory, pp. 392–403. Springer, Heidelberg (1996)

[KRS95] Karpinski, M., Rytter, W., Shinohara, A.: Pattern-matching for strings with short description. In: Galil, Z., Ukkonen, E. (eds.) CPM 1995. LNCS, vol. 937, pp. 205–214. Springer, Heidelberg (1995)

[Lif07] Lifshits, Y.: Processing compressed texts: A tractability border. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 228–240. Springer, Heidelberg (2007)

[LM05] Lohrey, M., Maneth, S.: The complexity of tree automata and XPath on grammar-compressed trees. In: Farré, J., Litovsky, I., Schmitz, S. (eds.) CIAA 2005. LNCS, vol. 3845. Springer, Heidelberg (2006)

[LMSS09] Lohrey, M., Maneth, S., Schmidt-Schauß, M.: Parameter reduction in grammar-compressed trees. In: de Alfaro, L. (ed.) FoSSaCS 2009. LNCS, vol. 5504, pp. 212–226. Springer, Heidelberg (2009)

[Loh06] Lohrey, M.: Word problems and membership problems on compressed words. SIAM Journal on Computing 35(5), 1210–1240 (2006)

[LR06] Lasota, S., Rytter, W.: Faster algorithm for bisimulation equivalence of normed context-free processes. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 646–657. Springer, Heidelberg (2006)

[LSSV04] Levy, J., Schmidt-Schauß, M., Villaret, M.: Monadic second-order unification is NP-complete. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 55–69. Springer, Heidelberg (2004)

[LSSV06a] Levy, J., Schmidt-Schauß, M., Villaret, M.: Bounded second-order unification is NP-complete. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 400–414. Springer, Heidelberg (2006)

[LSSV06b] Levy, J., Schmidt-Schauß, M., Villaret, M.: Stratified context unification is NP-complete. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS, vol. 4130, pp. 82–96. Springer, Heidelberg (2006)

[MM82] Martelli, A., Montanari, U.: An efficient unification algorithm. ACM Trans. on programming languages and systems 4(2), 258–282 (1982)

[MMS08] Maneth, S., Mihaylov, N., Sakr, S.: XML tree structure compression. DEXA, 243–247 (2008)

[MST97] Miyazaki, M., Shinohara, A., Takeda, M.: An improved pattern matching algorithm for strings in terms of straight-line programs. In: Hein, J., Apostolico, A. (eds.) CPM 1997. LNCS, vol. 1264, pp. 1–11. Springer, Heidelberg (1997)

[Pla95] Plandowski, W.: The Complexity of the Morphism Equivalence Problem for Context-Free Languages. PhD thesis, Department of Mathematics, Informatics and Mechanics, Warsaw University (1995)

[Rob65] Alan Robinson, J.: A machine oriented logic based on the resolution principle. J. of the ACM 12(1), 23–41 (1965)

[SS05] Schmidt-Schauß, M.: Polynomial equality testing for terms with shared substructures. Frank report 21, FB Informatik und Mathematik. Goethe-Univ. Frankfurt (November 2005)

Unification and Narrowing in Maude 2.4*

Manuel Clavel[1,2], Francisco Durán[3], Steven Eker[4], Santiago Escobar[5], Patrick Lincoln[4], Narciso Martí-Oliet[2], José Meseguer[6], and Carolyn Talcott[4]

[1] IMDEA Software, Madrid, Spain
[2] Universidad Complutense de Madrid, Spain
[3] Universidad de Málaga, Spain
[4] SRI International, CA, USA
[5] Universidad Politécnica de Valencia, Spain
[6] University of Illinois at Urbana-Champaign, IL, USA

Abstract. Maude is a high-performance reflective language and system supporting both equational and rewriting logic specification and programming for a wide range of applications, and has a relatively large worldwide user and open-source developer base. This paper introduces novel features of Maude 2.4 including support for unification and narrowing. Unification is supported in Core Maude, the core rewriting engine of Maude, with commands and metalevel functions for order-sorted unification modulo some frequently occurring equational axioms. Narrowing is currently supported in its Full Maude extension. We also give a brief summary of the most important features of Maude 2.4 that were not part of Maude 2.0 and earlier releases. These features include communication with external objects, a new implementation of its module algebra, and new predefined libraries. We also review some new Maude applications.

1 Introduction

Maude is a language and a system based on rewriting logic [7]. Maude modules are rewrite theories, while computation with such modules corresponds to efficient deduction by rewriting. Because of its logical basis and its initial model semantics, a Maude module defines a precise mathematical model. This means that Maude and its formal tool environment can be used in three, mutually reinforcing ways: as a declarative programming language, as an executable formal specification language, and as a formal verification system.

The first version of Maude was publicly released at the beginning of 1999 and presented at RTA'99 [5]; four years later, Maude 2.0 was introduced at RTA'03 [6]. The new and improved features since Maude 2.0 include: built-in AC unification; narrowing;

* M. Clavel has been partially supported by MICINN grants TIN2005-09207-C03-03 and TIN2006-15660-C02-01, and by CAM program S-0505/TIC/0407. F. Durán has been partially supported by MICINN grant TIN2008-03107 and Junta de Andalucía P06-TIC2250 and P07-TIC3184. S. Escobar has been partially supported by MICINN grant TIN2007-68093-C02-02, Integrated Action HA 2006-0007, and Generalitat Valenciana GVPRE/2008/113. P. Lincoln's effort partially supported by NSF grant CNS-0749931. N. Martí-Oliet has been partially supported by MICINN grant TIN2006-15660-C02-01 and CAM program S-0505/TIC/0407.

R. Treinen (Ed.): RTA 2009, LNCS 5595, pp. 380–390, 2009.
© Springer-Verlag Berlin Heidelberg 2009

object-message fairness and communication with external objects; a new implementation at the core level of its module algebra; new predefined libraries of parameterized data types; and a linear Diophantine equation solver.

Unification is built-in in Core Maude 2.4. Currently, narrowing is available in *Full Maude* [12,7], an extension of Maude written in Maude itself by taking advantage of its reflective capabilities. It has been used as a testbed for prototyping new features: parameterization [12], strategies [20], unification [7], and so on; and as a key component to build various formal tools by reflection (see Section 5).

There are several functional-logic programming languages based on narrowing (see, e.g., `http://www.informatik.uni-kiel.de/~mh/FLP/implementations.html`). How-ever, we are not aware of any other programming language supporting AC-narrowing, or combining narrowing with all the other features that Maude provides.

The releases of Maude since Maude 2.0 have added many other new features and improvements that cannot be described here. We refer the reader to the Maude documentation [8] for more details. The LNCS book on Maude [7] contains many additional examples and explanations, as well as information on applications and tools. However, the book only covers up to Maude 2.3, and therefore does not cover features like AC unification and narrowing. The Maude system, its documentation, and related papers and applications are available from its website at `http://maude.cs.uiuc.edu`.

2 Unification

Unification is a fundamental deductive mechanism used in many automated deduction tasks. It is also very important in combining the paradigms of functional programming and logic programming. Furthermore, in the context of Maude, unification can be very useful to reason not only about equational theories, but also about rewrite theories. In this section, we explain how order-sorted unification modulo frequently occurring equational axioms is currently supported in Maude 2.4.

Although the most general equational theories supported by Maude are membership equational theories, to obtain practical unification algorithms, allowing us to effectively compute the solutions of an equational unification problem, it is important to restrict ourselves to *order-sorted* equational theories. Furthermore, for an arbitrary set of equations no unification algorithm may be known; even if one is known, the number of solutions may be *infinite*. This suggests a hybrid approach, in which we take advantage of Maude's structuring of a module's equations into equational axioms Ax, such as associativity, and/or commutativity, and/or identity, and equations E, which are assumed to be confluent and terminating modulo Ax. We can then consider order-sorted equational theories of the form $(\Sigma, E \cup Ax)$ and decompose the $E \cup Ax$-unification problem into two problems: one of Ax-unification, and another of $E \cup Ax$-unification that uses an Ax-unification algorithm as a subroutine. The point is that *only* Ax-unification needs to be built-in at the level of Core Maude's C++ implementation for efficiency purposes. Instead, $E \cup Ax$-unification can then be implemented in Maude itself. Since the axioms Ax are well-known and unification algorithms exist for them, the task of building in efficient Ax-unification algorithms, although difficult, becomes manageable.

Unlike unsorted syntactic unification, which always either fails or has a single most general unifier, order-sorted syntactic unification is *not* unitary, that is, there is in

general no single most general unifier. What exists (if Σ is finite) is a finite minimal complete set of syntactic unifiers. For some commonly occurring theories having a unification algorithm, such as associativity of a binary function symbol, it is well-known that unification is not finitary. However, for other theories, such as commutativity (C) or associativity-commutativity (AC), unification is finitary, both when Σ is unsorted and order-sorted (and finite). Maude 2.4 provides an order-sorted Ax-unification algorithm for all order-sorted theories $(\Sigma, E \cup Ax)$ such that:

- the signature Σ is *preregular* modulo Ax [7];
- the axioms Ax associated to function symbols are as follows:
  - there can be arbitrary function symbols and constants with no attributes;
  - the `iter` equational attribute can be declared for some unary symbols;
  - the `comm` or `assoc comm` attributes can be declared for some binary function symbols, but no other equational attributes can be given for such symbols.

Explicitly excluded are theories with binary function symbols having either: (i) the `id:`, `left id:`, or `right id:` attributes; or (ii) the `assoc` attribute without the `comm` one; or (iii) a combination of (i) and (ii). The reason for excluding the `assoc` attribute without `comm` is the already-mentioned fact that associative unification is not finitary. The reason for excluding for the moment the `id:`, `left id:`, and `right id:` attributes is that they are *collapse equations* (one of the terms in the equation is a variable), requiring a more complex way of combining their unification algorithms. However, Ax-unification, where Ax includes such `id:`, `left id:`, and `right id:` attributes, is currently supported in Full Maude by narrowing (see Section 3).

If we give to Maude a unification problem in a functional module `fmod` $(\Sigma, E \cup Ax)$ `endfm`, then the equations E are ignored and we get a complete set of order-sorted unifiers modulo the theory (Σ, Ax). To deal with $E \cup Ax$-unification, other methods, that use the Ax-unification algorithm as a component, can later be defined (see Section 5).

Maude provides a unification command of the form:

`unify [`n`] in` *ModId* `:` t_1 `=?` t'_1 `/\ ... /\` t_k `=?` t'_k `.`

where $k \geq 1$, n is an optional argument providing a bound on the number of unifiers, and *ModId* is the name of the module or theory in which the unification takes place.

The use of a bound on the number of unifiers, as well as the use of the AC operator `_+_` in the predefined `NAT` module, plus the fact that even small AC-unification problems can generate a large number of unifiers are all illustrated by the following command:

```
Maude> unify [10] in NAT : X:Nat + X:Nat + Y:Nat =? A:Nat + B:Nat .
Solution 1
X:Nat --> #1:Nat + #2:Nat + #4:Nat
Y:Nat --> #3:Nat + #5:Nat
A:Nat --> #1:Nat + #1:Nat + #2:Nat + #3:Nat
B:Nat --> #2:Nat + #4:Nat + #4:Nat + #5:Nat
...
Solution 10
X:Nat --> #1:Nat + #2:Nat
Y:Nat --> #3:Nat
A:Nat --> #1:Nat + #1:Nat
B:Nat --> #2:Nat + #2:Nat + #3:Nat
```

Notice that in each assignment X `-->` t in a unifier, the variables appearing in the term t are always *fresh* variables of the form `#n:Sort`. Assuming that no bound on the number of unifiers is specified by the user, Maude will compute a *complete* set of order-sorted unifiers modulo Ax, for Ax a set of supported equational axioms. However, there is no guarantee that the computed set of unifiers is *minimal*, that is, some of the unifiers in the computed set may be redundant, since they could be obtained as instances (modulo Ax) of other unifiers in the set.

Order-sorted unification is NP-complete in general because Boolean algebra can be encoded as an order-sorted free theory signature and hence satisfiability can be reduced to an order-sorted free theory unification problem. In practice, reasonable performance can be obtained using a Binary Decision Diagram technique to compute sorts for free variables occurring in unsorted unifiers. Furthermore in the AC case, sort information can be pushed into the unsorted unification algorithm and used to prune the Diophantine basis and the choice of subsets drawn from such a basis [13].

The unification theory combination framework and AC unification algorithm are based on [4] while the Diophantine system solver used by the AC algorithm is based on [10]. The unification algorithm has been thoroughly tested (by S. Escobar and R. Sasse) using CiME [11] as an oracle, and has shown better average performance than CiME on the same problems.

Much of Maude's functionality is supported in its metalevel, so that it becomes available by reflection [7]. Unification is reflected in by the following descent functions:

```
op metaUnify : Module UnificationProblem Nat Nat ~> UnificationPair? .
op metaDisjointUnify :
   Module UnificationProblem Nat Nat ~> UnificationTriple? .
```

The key difference between `metaUnify` and `metaDisjointUnify` is that the latter assumes that the variables in the left- and right-hand unificands are to be considered *disjoint* even when they are not so, and it generates each solution to the given unification problem not as a single substitution, but as a *pair* of substitutions, one for left unificands and the other for right unificands. This functionality is very useful for applications, such as critical-pair checking or narrowing (see Section 3), where a disjoint copy of the terms or rules involved must always be computed before unification is performed.

Since it is convenient to reuse variable names from unifiers in new problems, for example in narrowing, this is allowed via the third argument, which is the largest number n appearing in a unificand metavariable of the form `#n:Sort`. Then the fresh metavariables in the computed unifiers will all be numbered from $n+1$ on.

Results are returned using the following constructors:

```
subsort UnificationPair < UnificationPair? .
subsort UnificationTriple < UnificationTriple? .
op {_,_} : Substitution Nat -> UnificationPair [ctor] .
op {_,_,_} : Substitution Substitution Nat -> UnificationTriple [ctor] .
```

The `Nat` component is the largest n occurring in a fresh `#`n`:Sort` metavariable. In this way, the next invocation of the function can use this parameter to make sure that the new variables generated are always fresh.

Examples illustrating the use of these metalevel functions can be found in [8].

3 Narrowing

Narrowing generalizes term rewriting by allowing free variables in terms (as in logic programming) and by performing unification instead of matching in order to (non–deterministically) reduce a term.

At each narrowing step, one must choose which subterm of the subject term, which rule of the specification, and which instantiation on the variables of the subject term and the rule's lefthand side is going to be considered. Given an order-sorted rewrite theory (Σ, Ax, R) where R is a set of unconditional rewrite rules such that the lefthand sides are non-variable terms and the rules are explicitly Ax-coherent [22], and Ax is a set of axioms such that a finitary Ax-unification procedure is available in Maude, the R,Ax-*narrowing* relation is defined as $t \leadsto_{\sigma,p,R,Ax} t'$ iff there is a non-variable position p of t, a (possibly renamed) rule $l \to r$ in R, and a unifier $\sigma \in Unif_{Ax}(t|_p, l)$ such that $t' = \sigma(t[r]_p)$. Full Maude supports a version of narrowing *with simplification*. That is, given an order-sorted rewrite theory $(\Sigma, Ax \cup E, R)$ where R and Ax are defined as above and E are the remaining equations, the combined relation $(\leadsto_{\sigma,p,R,Ax}; \to^!_{E,Ax})$ is defined as $t \leadsto_{\sigma,p,R,Ax}; \to^!_{E,Ax} t''$ iff $t \leadsto_{\sigma,p,R,Ax} t'$, $t' \to^*_{E,Ax} t''$, and t'' is E,Ax-irreducible. Note that this combined relation may be incomplete, i.e., given a reachability problem of the form $t \to^* t'$ and a solution σ (i.e., $\sigma(t) \to^*_{R,E \cup Ax} \sigma(t')$), the relation $\leadsto_{\sigma,p,R,Ax}; \to^!_{E,Ax}$ may not be able to find a more general solution. The reason is that the equations E should also be executed by narrowing instead of rewriting to ensure completeness under appropriate conditions (see [22] and Section 5).

The user can enter in Full Maude a search command of the form:

search [*n*,*m*] in *ModId* : t_1 *SearchArrow* t_2 .

where: n and m are optional arguments providing, respectively, a bound on the number of desired solutions and the maximum depth of the search; *ModId* is the module where the search takes place; t_1 is the starting *non-variable term*, which may contain variables; t_2 is the term specifying the pattern that has to be reached, with variables possibly shared with t_1; *SearchArrow* is an arrow indicating the form of the narrowing proof from t_1 until t_2 (~>1 for a narrowing proof consisting of exactly one step; ~>+ for a proof of one or more steps; ~>* for a proof of none, one, or more steps; and ~>! to indicate that only *strongly irreducible* final states are allowed, i.e., states that cannot be further narrowed).

Consider, for example, the following Petri-net-like specification of a vending machine to buy apples (a) or cakes (c) with dollars ($) and/or quaters (q):

```
(mod VENDING-MACHINE is
   sorts Coin Item Marking Money State .
   subsort Coin < Money .
   op __ : Money Money -> Money [assoc comm] .
   subsort Money Item < Marking .
   op __ : Marking Marking -> Marking [assoc comm] .
   op <_> : Marking -> State .
   ops $ q : -> Coin [format (r! o)] .
   ops a c : -> Item [format (b! o)] .
   var M : Marking .
   rl [buy-c] : < $ > => < c > .
```

```
  rl [buy-c] : < M $ > => < M c > .
  rl [buy-a] : < $ > => < a q > .
  rl [buy-a] : < M $ > => < M a q > .
  rl [change]: < q q q q > => < $ > .
  rl [change]: < M q q q q > => < M $ > .
endm)
```

We can use the narrowing search command to answer the question: *Is there any combination of one or more coins that returns exactly an apple and a cake?* This is done by searching for states that have a variable of sort `Money` instead of sort `Marking` at the starting term and match a corresponding pattern at the end.

```
Maude> (search [,4] in VENDING-MACHINE : < M:Money > ~>* < a c > .)
Solution 1
M:Money --> $ q q q
Solution 2
M:Money --> q q q q q q q
```

Note that we must restrict the depth, because narrowing does not terminate for this reachability problem even though the above two solutions are indeed the *only* solutions.

Narrowing-based reachability analysis is also available at the metalevel by using the following `metaNarrowSearch` function.

```
op metaNarrowSearch :
  Module Term Term Substitution Qid Bound Bound -> ResultTripleSet .
```

If a non-identity substitution is provided in the fourth argument, then any computed substitution must be an instance of the provided one, i.e., we can restrict the computed narrowing sequences to some concrete shape. The `Qid` metarepresents the appropriate search arrow, similar to the `metaSearch` command (see [8, Section 11.4.6]). For the bounds, the first one is the number of computed solutions, and the second one is the maximum length of the narrowing sequences, i.e., the depth of the narrowing tree.

Unification modulo identity: The `id-unify` command. As described in Section 2, Maude 2.4 provides an order-sorted Ax-unification algorithm for all order-sorted theories $(\Sigma, E \cup Ax)$ such that Σ is preregular and Ax can include any combination of equational axioms for a function symbol *except* the `id:`, `left id:`, `right id:`, and `assoc` without `comm`. If a theory (Σ, Ax) contains the `id:`, `left id:`, or `right id:` attributes in Ax (but *not* `assoc` without `comm`), we can perform unification modulo Ax as follows:

1. we decompose Ax into a disjoint union $Ax = \widetilde{Ax} \cup Ids$, where $\widetilde{Ax}$ does not contain any `id:`, `left id:`, or `right id:` attribute, and Ids is the set of such extra attributes;
2. we define the rewrite theory $(\Sigma, \widetilde{Ax}, \overrightarrow{Ids})$ where $\overrightarrow{Ids}$ contains the obvious rules for each of the equational identity attributes, that is:
   - if f has an `id:` attribute in Ax and $[s]$ is the top sort of f (which we can identify with the kind), we add rules $f(x,e) \to x$ and $f(e,x) \to x$ into $\overrightarrow{Ids}$, where x is a variable of sort $[s]$ and e is the identity symbol (if f has also the `comm` attribute, only one such rule is needed);

- Likewise, for f with the `left id:` (resp. `right id:`) attribute, we add the rule $f(e,x) \rightarrow x$ (resp. $f(x,e) \rightarrow x$) into $\overrightarrow{Ids}$.

3. based on the idea of "variants" in [9], for the Ax-unification problem $t \stackrel{?}{=} t'$, we compute the variants of t and the variants of t' using narrowing modulo $\widetilde{Ax}$ within the theory $(\Sigma, \widetilde{Ax}, \overrightarrow{Ids})$ and perform $\widetilde{Ax}$-unification pairwise among all the variants of t and t' (see [17] for details).

The Full Maude `id-unify` command implements the above Ax-unification procedure (with $Ax = \widetilde{Ax} \cup Ids$) using variants.[1] Given a module or theory *ModId* having a set Ax of equational axioms for the signature Σ such that Σ does not include symbols with `assoc` without `comm` attributes in Ax, Full Maude provides a unification command for Ax-unification of the form:

```
id-unify in ModId : t =? t' .
```

where only one unification problem is admitted, in contrast to the `unify` command, and such that no limit to the number of unifiers can be specified.

The procedure for equational Ax-unification, where Ax can contain any Maude equational attribute except `assoc` without `comm`, is also available at the metalevel thanks to the `metaACUUnify` function.

```
op metaACUUnify : Module Term Term -> SubstitutionSet .
```

4 Other Available Features

In this section we briefly mention some of the other features introduced in Maude since Maude 2.0. More details can be found in the Maude documentation [8,7].

Object-message fairness and external objects. Distributed systems can be modeled as multisets of entities, coupled by some suitable communication mechanism. In object-based distributed systems, the entities are objects, each with a unique identity, and the communication mechanism is message passing. Maude 2.4 supports the modeling of such systems by providing a predefined `CONFIGURATION` module and an *object-message fair* rewriting strategy that is well suited for executing object system configurations. It also supports *external objects*, so that objects inside a Maude configuration can interact with different kinds of objects outside it. The external objects directly supported are internet *sockets*, but through them it is possible to interact with other external objects.

Module algebra. As in other languages in the Clear/OBJ tradition, the abstract syntax of Maude specifications can be seen as given by *module expressions*, defining a new module out of previously defined modules by combining or modifying them according to a specific set of operations. Maude 2.4 supports module operations for summation, renaming, and instantiation of parameterized modules. Theories, parameterized modules, and views are the basic building blocks of *parameterized* programming.

[1] Of course, this is less efficient than built-in Ax-unification. However, a number of useful applications can be supported in practice even with `id-unify` (see Section 5).

New predefined libraries. Maude has a standard library of predefined modules. In addition to predefined modules providing commonly used data types, such as Booleans, numbers, strings, and quoted identifiers, that were already available in Maude 2.0, the following modules are predefined. The `RANDOM` module provides a pseudo-random number generator, and the system module `COUNTER` a "counter" that can be used to generate new names. These modules can be used together, e.g., to specify *probabilistic models* in Maude [7]. For certain applications, it is convenient to have a predefined specification for machine integers instead of the arbitrary size integers provided by the `INT` module. The parameterized module `MACHINE-INT` takes a bit-width parameter $n \geq 2$, which must be a power of 2, and defines machine integer operations. For parameterized programming, several functional theories like `TRIV`, `DEFAULT`, `STRICT-WEAK-ORDER`, `TOTAL-PREORDER`, and `TOTAL-ORDER` are predefined. Also predefined are the modules `LIST`, `SET`, `LIST*`, and `SET*`. The `WEAKLY-SORTABLE-LIST` module, parameterized by `STRICT-WEAK-ORDER`, specifies a *stable* version of the *mergesort* algorithm, and the `SORTABLE-LIST` module sorts lists with respect to the `TOTAL-ORDER` theory.

5 Some Applications

In this section we review briefly some Maude applications that have been or can be developed, particularly with the new unification and narrowing infrastructure.

Narrowing-based unification. If we have a dedicated algorithm to solve unification problems in an order-sorted theory (Σ, Ax), then we can use it as a component to obtain a unification algorithm for theories of the form $(\Sigma, E \cup Ax)$, provided the equations E are coherent, confluent and terminating modulo Ax [18]. We just need to add to $(\Sigma, E \cup Ax)$ a new constant tt, a binary symbol eq, and equations of the form $eq(x,x) = tt$ (one for each top sort in Σ, with x of that top sort). Then we can reduce an $E \cup Ax$-unification problem $t \stackrel{?}{=} t'$ to the narrowing reachability problem $eq(t,t') \rightsquigarrow^* tt$ modulo Ax in the theory extending $(\Sigma, E \cup Ax)$ with these new operators, and equations.

The computation of $E \cup Ax$-unifiers by narrowing modulo Ax yields a complete but in general infinite set of $E \cup Ax$-unifiers. When $Ax = \emptyset$, sufficient conditions are known ensuring termination of the basic narrowing strategy (see, e.g., [19,1]), and therefore yielding a finite complete set of $E \cup Ax$-unifiers. However, for axioms Ax such as AC, it is well-known that narrowing modulo AC "almost never terminates" and, furthermore, that basic narrowing is incomplete [25,9]. Based on the idea of "variants" in [9], a complete narrowing strategy modulo Ax called *variant narrowing* has been proposed in [17]. Furthermore, in [16] sufficient checkable conditions on $(\Sigma, E \cup Ax)$ have been given ensuring that the $E \cup Ax$-unification algorithm provided by variant narrowing modulo Ax is finitary, even though variant narrowing modulo Ax may still not terminate in spite of such conditions. A Maude-based narrowing library that uses the current built-in unification algorithm as a component has been developed by S. Escobar.

Symbolic reachability analysis in rewrite theories. A rewrite theory, say $\mathcal{R} = (\Sigma, E \cup Ax, R)$, specified in Maude as a system module, describes a concurrent system whose states are $E \cup Ax$-equivalence classes of ground terms, and whose local concurrent transitions are specified by the rules R. When formally analyzing the properties of $\mathcal{R}$, an

important problem is ascertaining for specific patterns t and t' the *symbolic reachability problem* $(\exists X)\, t \longrightarrow^* t'$ with X the set of variables appearing in t and t'. As shown in [22], provided the rewrite theory $\mathcal{R} = (\Sigma, E \cup Ax, R)$ is *topmost* (that is, all rewrites take place at the root of a term), or, as in the case of *AC* rewriting of object-oriented systems, $\mathcal{R}$ is "essentially topmost," and the rules R are coherent with E modulo Ax, narrowing with the rules R modulo the equations $E \cup Ax$ gives a constructive, *sound, and complete* method to solve reachability problems of the form $(\exists X)\, t \longrightarrow^* t'$.

Of course, narrowing with R modulo $E \cup Ax$ requires performing $E \cup Ax$-unification at each narrowing step, which as explained above can itself be performed by narrowing with the equations E modulo Ax, provided E is coherent, confluent, and terminating modulo Ax. Therefore, in performing symbolic reachability analysis in a rewrite theory $\mathcal{R} = (\Sigma, E \cup Ax, R)$ there are usually *two* levels of narrowing and *two* levels of unification: narrowing with R modulo $E \cup Ax$ for reachability, and narrowing with E modulo Ax for unification modulo $E \cup Ax$. This is exactly the approach taken in the Maude-NPA protocol analyzer [14], where cryptographic protocols are formally specified as rewrite theories of the form $\mathcal{R} = (\Sigma, E \cup Ax, R)$, and the formal reachability analysis is performed in a *backwards* way, from an attack state to an initial state. This just means that we perform standard (forwards) reachability analysis with the rewrite theory $\mathcal{R}^{-1} = (\Sigma, E \cup Ax, R^{-1})$, where $R^{-1} = \{r \longrightarrow l \mid (l \longrightarrow r) \in R\}$. The equational theory $E \cup Ax$ typically specifies the algebraic properties of the cryptographic functions used in the given protocol, for example, public key encryption and decryption, exclusive or, modular exponentiation, and so on, which often have the finite variant property [9].

Solving a symbolic reachability problem $(\exists X)\, t \longrightarrow^* t'$ corresponds to *falsifying* an *invariant*, namely, that all states reachable from t are in the complement of the instances of t'. The paper [15] shows how narrowing can be used to perform a more general *symbolic model checking*, not just for invariants, but for temporal logic formulas.

Building Formal Tools Reflectively in Maude. Another important application area is the development of formal tools by reflection. This can be done directly in Core Maude using the `META-LEVEL` module, or in Full Maude as a language extension.The Maude book [7] describes many such tools: the Maude inductive theorem prover, Church-Rosser checker, coherence checker, sufficient completeness checker, termination tool, real-time Maude tool, and several others. Some recent new tools are the Maudeling and MOMENT-2 tools, for formal specification and analysis in model-based software engineering [23,3], the already mentioned Maude-NPA [14], and a model checker for the linear temporal logic of rewriting [2]. With metalevel support for unification and narrowing, new formal tools can be built in the near future.

The Rewriting Logic Semantics Project. An important and very active area of Maude applications is based on the idea of giving formal semantics to a programming language $\mathcal{L}$ as a rewrite theory $\mathcal{R}_{\mathcal{L}} = (\Sigma_{\mathcal{L}}, E_{\mathcal{L}}, R_{\mathcal{L}})$, where $\Sigma_{\mathcal{L}}$ defines the syntax and the semantic types of $\mathcal{L}$, $E_{\mathcal{L}}$ the deterministic semantics, and $R_{\mathcal{L}}$ the concurrent semantics (see [21,24]). Specifying $\mathcal{R}_{\mathcal{L}}$ in Maude as a system module yields not only an $\mathcal{L}$-interpreter, but also an $\mathcal{L}$-model checker that can perform sophisticated program analysis.

References

1. Alpuente, M., Escobar, S., Iborra, J.: Modular termination of basic narrowing. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 1–16. Springer, Heidelberg (2008)
2. Bae, K., Meseguer, J.: A rewriting-based model checker for the linear temporal logic of rewriting. In: Procs. of RULE 2008. ENTCS (to appear) (2008)
3. Boronat, A., Meseguer, J.: An Algebraic semantics for MOF. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 377–391. Springer, Heidelberg (2008)
4. Boudet, A., Contejean, E., Devie, H.: A new AC unification algorithm with an algorithm for solving systems of diophantine equations. In: Procs. of LICS 1990, pp. 289–299 (1990)
5. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.: The Maude system. In: Narendran, P., Rusinowitch, M. (eds.) RTA 1999. LNCS, vol. 1631, pp. 240–243. Springer, Heidelberg (1999)
6. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, J.: The Maude 2.0 system. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 14–29. Springer, Heidelberg (2003)
7. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007)
8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: Maude Manual (v. 2.4), SRI Intl. & U. of Illinois at Urbana-Champaign (October 2008), `http://maude.cs.uiuc.edu`
9. Comon-Lundh, H., Delaune, S.: The finite variant property: how to get rid of some algebraic properties. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 294–307. Springer, Heidelberg (2005)
10. Contejean, E., Devie, H.: An efficient incremental algorithm for solving systems of linear diophantine equations. Information and Computation 113(1), 143–172 (1994)
11. Contejean, E., Marché, C., Urbain, X.: CiME 3 (2004), `http://cime.lri.fr/`
12. Durán, F., Meseguer, J.: Maude's module algebra. Sci. Comp. Progr. 66(2), 125–153 (2007)
13. Eker, S.: Unification in Maude. Talk at the Protocol eXchange Seminar, Naval Postgraduate School (January 2007), `http://maude.cs.uiuc.edu/talks/eker-unification.pdf`
14. Escobar, S., Meadows, C., Meseguer, J.: A rewriting-based inference system for the NRL Protocol Analyzer and its meta-logical properties. Theor. Comput. Sci. 367(1-2), 162–202 (2006)
15. Escobar, S., Meseguer, J.: Symbolic model checking of infinite-state systems using narrowing. In: Baader, F. (ed.) RTA 2007. LNCS, vol. 4533, pp. 153–168. Springer, Heidelberg (2007)
16. Escobar, S., Meseguer, J., Sasse, R.: Effectively checking the finite variant property. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 79–93. Springer, Heidelberg (2008)
17. Escobar, S., Meseguer, J., Sasse, R.: Variant narrowing and equational unification. In: Procs. of WRLA 2008, pp. 88–102. ENTCS (2008)
18. Jouannaud, J.-P., Kirchner, C., Kirchner, H.: Incremental construction of unification algorithms in equational theories. In: Díaz, J. (ed.) ICALP 1983. LNCS, vol. 154, pp. 361–373. Springer, Heidelberg (1983)
19. Hullot, J.-M.: Canonical forms and unification. In: Bibel, W. (ed.) CADE 1980. LNCS, vol. 87, pp. 318–334. Springer, Heidelberg (1980)
20. Martí-Oliet, N., Meseguer, J., Verdejo, A.: Towards a strategy language for Maude. In: Procs. of WRLA 2004. ENTCS, vol. 117, pp. 417–441 (2005)
21. Meseguer, J., Roşu, G.: The rewriting logic semantics project. Theor. Comput. Sci. 373(3), 213–237 (2007)

22. Meseguer, J., Thati, P.: Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. High.-Ord. Symb. Comp. 20(1-2), 123–160 (2007)
23. Rivera, J.E., Vallecillo, A.: Adding behavioral semantics to models. In: Procs. of EDOC 2007, pp. 169–180 (2007)
24. Şerbănuţă, T.F., Roşu, G., Meseguer, J.: A rewriting logic approach to operational semantics. Information and Computation. (available online December 6, 2008) (in press)
25. Viola, E.: E-unifiability via narrowing. In: Restivo, A., Ronchi Della Rocca, S., Roversi, L. (eds.) ICTCS 2001. LNCS, vol. 2202, pp. 426–438. Springer, Heidelberg (2001)

Author Index

Aoto, Takahito 93
Avanzini, Martin 48

Baader, Franz 350
Baudet, Mathieu 148
Bursuc, Sergiu 133

Clavel, Manuel 380
Comon-Lundh, Hubert 133
Cortier, Véronique 148

de Falco, Marc 209
Delaune, Stéphanie 148
de Vrijer, Roel 270
Durán, Francisco 380
Duval, Dominique 194

Echahed, Rachid 194
Eker, Steven 380
Endrullis, Jörg 270, 305
Escobar, Santiago 380

Falke, Stephan 32
Fuhs, Carsten 32

Gascón, Adrià 365
Giesl, Jürgen 32
Godoy, Guillem 63, 365
Goré, Rajeev 103
Gramlich, Bernhard 285

Hendriks, Dimitri 305

Jacquemard, Florent 63

Kahrs, Stefan 179
Ketema, Jeroen 239
Kimura, Daisuke 224
Korp, Martin 295

Lincoln, Patrick 380
López-Fraguas, Francisco Javier 320

Martí-Oliet, Narciso 380
Meseguer, José 380
Middeldorp, Aart 295
Morawska, Barbara 350
Moser, Georg 48, 255

Plücker, Martin 32
Prost, Frédéric 194

Rodríguez-Hortalá, Juan 320

Sánchez-Hernández, Jaime 320
Schernhammer, Felix 285
Schmidt-Schauß, Manfred 365
Schnabl, Andreas 255
Schneider-Kamp, Peter 32
Schubert, Aleksy 78
Seidl, Helmut 118
Simonsen, Jakob Grue 335
Sternagel, Christian 17, 295

Talcott, Carolyn 380
Tatsuta, Makoto 224
Thiemann, René 17
Tiu, Alwen 103
Toyama, Yoshihito 93

Verma, Kumar Neeraj 118

Waldmann, Johannes 1, 270

Yoshida, Junichi 93

Zankl, Harald 295
Zantema, Hans 164