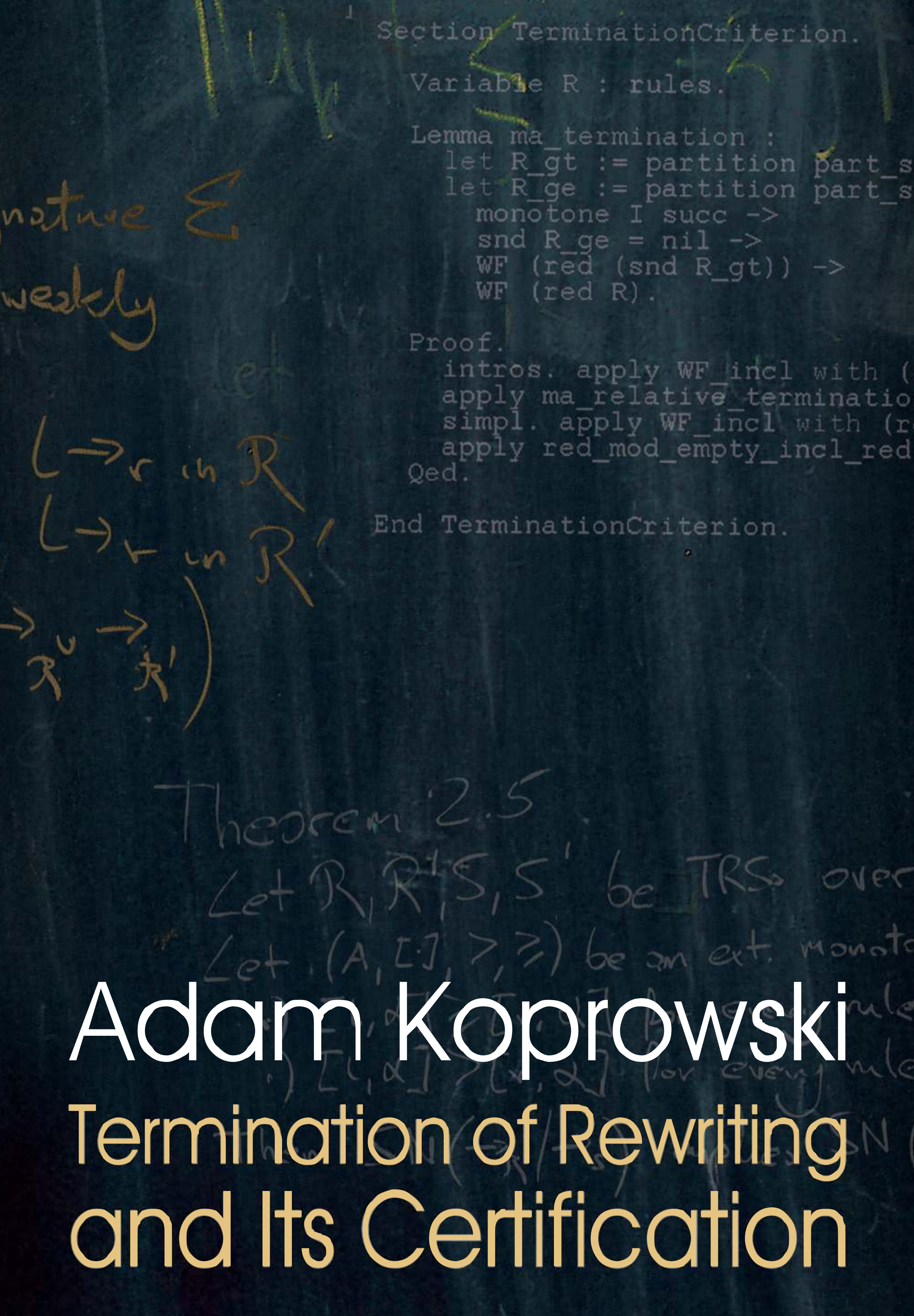
Termination of rewriting and its certification

DOI:
10.6100/IR637480

Document status and date:
Published: 01/01/2008

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)



Adam Koprowski

Termination of Rewriting and Its Certification

Termination of Rewriting and Its Certification

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op donderdag 25 september 2008 om 16.00 uur

door

Adam Wojciech Koprowski

geboren te Warschau, Polen

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. H. Zantema
en
prof.dr.ir. J.F. Groote

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Koprowski, Adam

Termination of Rewriting and Its Certification / Koprowski Adam. -
Eindhoven : Technische Universiteit Eindhoven, 2008.
Proefschrift. - ISBN 978-90-386-1377-2
NUR 993
CR Subject Classification: F.3.1, F.4.1, F.4.2
Subject headings: theoretical computer science / programming ; formal methods / software verification

Eerste promotor: prof.dr. H. Zantema (Eindhoven University of Technology)

Tweede promotor: prof.dr.ir. J.F. Groote (Eindhoven University of Technology)

Kerncommissie:

prof.dr. H. Geuvers (Radboud University Nijmegen)

prof.dr. J.W. Klop (VU University Amsterdam)

prof.dr. A. Middeldorp (University of Innsbruck)

Netherlands Organisation for Scientific Research

The work in this thesis is supported by Netherlands Organization for Scientific Research (NWO) in the framework of the project "Verification and Rewriting" (612.000.310).

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

IPA dissertation series 2008-24

Printing: Eindhoven University Press

Cover design: D. Kwoka

Contents

Preface

The journey leading to this thesis started for me four years ago. After studying at the Warsaw University I spent the last year of my studies on an exchange program at the Free University Amsterdam. It was a wonderful year and it was there that I met Femke van Raamsdonk, who later became the supervisor of my Master thesis. While I was approaching the end of this stage and had no good idea of where I wanted to go from there, it was she who suggested me to try to continue with Ph.D. studies and helped me to find a position, effectively setting me on track leading to this manuscript. For that my most sincere thanks go to her.

I am greatly indebted to Hans Zantema who offered me this position and who was my daily supervisor. For these past four years he has always been there when I needed his help and I could be sure that whenever I would present him my ideas, regardless of their merit or lack thereof, he would listen to them carefully and share his comments with me. Our numerous discussions and the enthusiasm he put into them were a constant source of inspiration. His carefully chosen level of supervision allowed me to gradually perform independent research. For all these reasons I would like to express my best thanks to him.

I am also grateful to Jan Friso Groote, the head of the Design and Analysis of Systems group, and to all the colleagues from the group, whose kindly atmosphere allowed me not only to finish the PhD in the group, but also to enjoy it along the way.

The very special thanks go to MohammadReza Mousavi. During my first years in Eindhoven he was a great office-mate. He was, kindly and patiently, showing me the ropes, concerning doing research as well as some more mundane matters. His invaluable help extended even into the creation of this manuscript as he was kind enough to share the template of his thesis, making the editing process so much easier for me. Mohammad, I am proud to call you my friend and the only thing I would not like to do with you is to go together for holidays ;-).

I would like to thank Herman Geuvers, Jan Friso Groote, Jan Willem Klop, Aart Middeldorp and Hans Zantema. As the members of my thesis committee, by reading the draft and sharing their useful comments they greatly helped to improve this manuscript.

Apart from my supervisor, Hans Zantema, during the previous years I had the pleasure to work together with Frédéric Blanqui, Aart Middeldorp and Johannes Waldmann. Those collaborations, and my short visits to, respectively, Nancy, Innsbruck and Leipzig, taught me a lot and for that my most sincere thanks. I am also greatly indebted to Jürgen Giesl for his kind support along the way.

I am thankful to the T$_\mathsf{T}$T team in Innsbruck for many productive discussions and for sharing their experience with the tool, which helped me with the development of TPA. Christian Sternagel deserves a special mention — I still remember the lovely lasagna he prepared during my one week stay in Innsbruck and now he was kind enough to read this manuscript and share his useful comments.

Dawid Kwoka designed the cover of this manuscript on a very short notice. Thanks for that.

Writing this thesis was a long and, sometimes, painful process. I mentioned a number of people that helped me to a great extent when it comes to the content. But I would not be able to finish this undertaking without a great support of a number of people. My words of gratitude go to my parents Lucyna and Krzysztof, my sister Ewa, and my friends; among which Agnieszka Maliszewska, Natalia Kryt, Łukasz Kuciński, Karol Osłowski and Łukasz Strzelec deserve a special mention.

The last but certainly not the least, I would like to thank my girlfriend, Alexandra. Words cannot express how much I am grateful for all her love and support. Alexandra, you make me a happy man and for that I would like to dedicate this thesis to you.

List of Notations

Notation	Explanation	Reference
$\to^*$	reflexive transitive closure of $\to$	Page 11
$\to^+$	transitive closure of $\to$	Page 11
$\to / \tilde{\to}$	$\to$ relative to $\tilde{\to}$	Def. 1.2
$\mathrm{NF}(\to)$	normal forms of $\to$	Def. 1.3
$\mathrm{NF}(\mathcal{I}, \to)$	normal forms of $\to$ reachable from $\mathcal{I}$	Def. 1.3
$\mathrm{SN}(\to)$	termination of $\to$	Def. 1.4
$\mathrm{SN}(\mathcal{I}, \to)$	termination of $\to$ on a set of states $\mathcal{I}$	Def. 1.4
$(>_1, \ldots, >_n)_{lex}$	lexicographic extension of n relations	Def. 1.5
$\mathcal{W}_<^{Acc}$	accessible part of a set $\mathcal{W}$	Def. 1.7
$\mathrm{arity}(f)$	arity of f	Def. 1.9
$\mathcal{T}(\mathcal{F}, \mathcal{V})$	the set of terms over a signature $\mathcal{F}$ and a set of variables $\mathcal{V}$	Def. 1.10
$\rhd, \unrhd$	subterm relations on terms	Def. 1.10
$\mathcal{V}\mathrm{ar}(t)$	the set of variables occurring in a term t	Def. 1.10
$\mathcal{F}\mathrm{un}(t)$	the set of function symbols occurring in a term t	Def. 1.10
$\mathrm{root}(t)$	the root of a term t	Def. 1.10
$\mathcal{R}_f$	the subset of a TRS $\mathcal{R}$ consisting of rules with f as root	Def. 1.12
$\mathcal{R}_\succ$	the subset of a TRS $\mathcal{R}$ oriented by relation $\succ$	Def. 1.12
$\to_\mathcal{R}$	the rewrite relation of a TRS $\mathcal{R}$	Def. 1.15
$\xrightarrow{\epsilon}_\mathcal{R}$	the top rewrite relation of a TRS $\mathcal{R}$	Def. 1.15
$\mathrm{SN}(\to_\mathcal{R}), \mathrm{SN}(\mathcal{R})$	termination of a TRS $\mathcal{R}$	Def. 1.17

Notation	Explanation	Reference
$\mathrm{SN}(\mathcal{R}/\mathcal{S})$ $\mathrm{SN}(\rightarrow_{\mathcal{R}} / \rightarrow_{\mathcal{S}})$	termination of $\mathcal{R}$ relative to $\mathcal{S}$	Def. 1.17
$\mathcal{SN}$	the set of all terminating terms	Def. 1.17
$\mathcal{T}_\infty$	the set of minimal non-terminating terms	Def. 1.17
$(A, \{f_A\}_{f\in\mathcal{F}})$	a $\mathcal{F}$-algebra	Def. 1.18
$[t]_\alpha$	interpretation of a term t with a valuation α	Def. 1.18
$\mathcal{C}_\mathcal{E}$	$\mathcal{C}_\mathcal{E}$-compatibility	Def. 1.22
$(A, \{f_A\}_{f\in\mathcal{F}}, >, \gtrsim)$	weakly monotone $\mathcal{F}$-algebra	Def. 1.25
$\gtrsim_\mathcal{A}$, $>_\mathcal{A}$	extensions of weakly monotone algebra orders	Def. 1.26
$\mathcal{P}_\mathbb{N}(\vec{x})$	a polynomial with natural number coefficients over a vector of variables $\vec{x}$	Page 19
$\succ^{\mathsf{RPO}}$	recursive path order (RPO)	Def. 1.29
$\tau(f)$	status of a function symbol f for RPO	Def. 1.29
L_f	a set of labels for semantic labeling	Def. 1.33
ℓ_f	a set of interpretations for semantic labeling	Def. 1.33
$\mathcal{F}_{\mathsf{lab}}$	labeled signature	Def. 1.34
lab_α	labeling function	Def. 1.35
$\mathcal{D}\mathrm{ec}_\mathcal{R}$	decreasing TRS	Def. 1.36
$\mathcal{R}_{\mathsf{lab}}$	labeled TRS	Def. 1.37
$\rhd_\mathrm{d}$	dependability relation for predictive labeling	Def. 1.40
$\mathcal{G}_\ell(\mathcal{R}), \mathcal{U}(\ell)$	usable rules for labeling ℓ	Def. 1.41
$\mathcal{D}_\mathcal{R}$	defined symbols of a TRS $\mathcal{R}$	Def. 1.45
$\mathcal{F}^\sharp$	signature for dependency pairs	Def. 1.45
$\mathrm{DP}(\mathcal{R})$	dependency pairs of $\mathcal{R}$	Def. 1.45
$\pi(t)$	an argument filtering π applied to a term t	Def. 1.51
$\succ^\pi$	relation $\succ$ lifted to terms filtered with an argument filtering π	Def. 1.51
$\mathcal{U}_\pi(\mathcal{P}, \mathcal{R})$	the set of usable rules for a dependency problem $(\mathcal{P}, \mathcal{R})$ modulo π	Def. 1.52
$M[i, j]$	element at position (i, j) of a matrix M	
$\vec{x}[i]$	element at position i of a vector $\vec{x}$	

Notation	Explanation	Reference
$\oplus$	semiring addition	Page 35
$\otimes$	semiring multiplication	Page 35
$\mathcal{A}_{\mathbb{N}}$	arctic naturals	Page 35
$\mathcal{A}_{\mathbb{Z}}$	arctic integers	Page 35
$\gg$	order on arctic naturals/integers	Page 36
$\gg_\lambda, \geqslant_\lambda$	orders on arctic linear functions	Def. 2.13
ϕ_f	a label synthesis function for f	Def. 3.4
pd	precedence description	Def. 3.4
$[\alpha]^*_{\mathcal{A}}(t), \mathsf{lab}^*_\alpha(t)$ $\alpha^*_\sigma, \sigma_{\mathsf{lab}^*_\alpha}$	auxiliary definitions for predictive labeling over natural numbers	Theorem 3.14
$\ulcorner \ldots \urcorner$	encoding of some construct into propositional satisfiability	
$\omega_\circ(\cdots)$	SAT formula for some construct indicated by the subscript	
$\succ_{\mathcal{F}_{\mathsf{lab}}}$	encoding of precedence comparison for predictive labeling	Def. 3.17
$\mathrm{Live}(\mathcal{I}, \rightarrow, \mathcal{G})$	liveness property	Def. 5.1
$\rightarrow_{\mathcal{G}}$	restricted reduction relation for liveness	Def. 5.1
$\mathrm{Live}(\mathcal{I}, \rightarrow, \tilde{\rightarrow}, \mathcal{G})$	liveness with a fairness property	Def. 5.3
$\mathcal{T}_{\mathrm{top}}(\mathcal{F}, \mathcal{V})$	the set of top terms	Def.5.5
$\mathrm{LS}(\mathcal{R}, p)$	liveness transformation	Def.5.6
$\mathrm{LT}(\mathcal{R}), \mathrm{LT}^=(\mathcal{R}_\sim, P)$	liveness with fairness transformation	Def. 5.8
$\mathbb{M}_A$	the set of finite multisets	Def. 7.1
$\{\{a, \ldots\}\}$	multiset containing elements a, ...	
$\dot{-}$	bounded subtraction on natural numbers	Table 7.2
$\rhd_{mul}$	multiset reduction	Def. 7.2
$>_{mul}$	multiset extension of a relation	Def. 7.2
$>_{MUL}$	multiset extension of a relation	Def. 7.3
$\mathbb{M}^{Acc}_{\lhd}$	multisets accessible with respect to $\lhd_{mul}$	Def. 7.3
$\lambda^{\rightarrow}$	simply typed lambda calculus	Chapter 8
$\mathcal{T}_{\mathcal{S}}$	simple types	Def. 8.1
$\mathcal{E}nv$	environments	Def. 8.3
$\mathcal{P}t$	un-typed terms	Def. 8.4

Notation	Explanation	Reference
$\Gamma \vdash t : A$	a typing judgement	Def. 8.5
Λ	typed terms	Def. 8.6
$\overline{\Lambda}$	fully applied terms	Def. 8.7
$\Gamma \rightleftharpoons \Delta$	environment compatibility	Def. 8.9
$\mathrm{Vars}(t)$	free variables of a term t	Def. 8.15
$\sqsubset, \sqsubseteq$	subterm relations for simply typed terms	Def. 8.16
$t\vert_p$	subterm of t at position p	Def. 8.21
$t[u]_p$	replacing u in t at position p	Def. 8.22
$t{\uparrow}^n_k, t{\uparrow}^n$	term lifting	Def. 8.24
$\Gamma{\uparrow}^n_k, \Gamma{\uparrow}^n$	environment lifting	Def. 8.25
$t{\downarrow}^n_k, t{\downarrow}^n$	term lowering	Def. 8.27
$\Gamma{\downarrow}^n_k, \Gamma{\downarrow}^n$	environment lowering	Def. 8.28
$Dom(\tau)$	substitution domain	Def. 8.30
$Ran(\tau)$	substitution range	Def. 8.30
$\tau \backslash \mathcal{X}$	substitution with restricted domain	Def. 8.30
$t\tau$	substitution application	Def. 8.31
$\tau{\uparrow}^n$	substitution lifting	Def. 8.32
$\Gamma \overset{\Phi}{\approx} \Delta$	environment convertibility	Def. 8.38
$t \overset{\Phi}{\approx} u$	un-typed term convertibility	Def. 8.39
$t \sim u$	term convertibility	Def. 8.40
$\tau \overset{\Phi}{\sim} \delta$	substitution convertibility	Def. 8.41
$\rightarrow_\beta$	beta reduction of the $\lambda^{\rightarrow}$	Def. 8.43
$\mathcal{P}t_a$	algebraic terms	Def. 9.2
$\mathbb{C}_\delta$	the set of computable terms of type δ	Def. 9.7
$\succ^{\mathsf{HORPO}}$	higher-order recursive path ordering (HORPO)	Def. 9.15
$\succ\!\!\succ^{\mathsf{HORPO}}$	auxiliary ordering on terms for HORPO	Def. 9.15
$\succ^{\mathsf{HORPO}}_{MUL}$	a multiset extension of HORPO	Def. 9.15
$\rightsquigarrow$	union of HORPO and β-reduction	Page 166

Introduction to the Thesis

The Subject Matter

Rewriting has been nicely and succinctly summarized in the following passage in [TeR03]:

> *Rewriting is the theory of stepwise, or discrete, transformations of objects, as opposed to continuous transformations of objects. Since computations in computer science are typically stepwise transformations of objects, rewriting is therefore a foundational theory of computing.*

Term rewriting is a specialization of rewriting where the objects under consideration are terms. One of the most important problems within the world of term rewriting is that of *termination*. In the main annual conference on term rewriting, *RTA (Rewriting Techniques and Applications)*, termination is usually the most represented subject. Since 1993, a *Workshop on Termination* is also being organized on a regular basis.

Termination states that given some computation rules no infinite computation sequence can occur and it is required in order to establish *total correctness* of a program/algorithm. In term rewriting computation is expressed by means of a *term rewriting system (TRS)*. As term rewriting is Turing-complete, the termination problem for TRSs is undecidable.

Many methods have been devised to prove termination of term rewriting, some of which will be presented in the coming chapters. Most of the work in this area concentrates on proving termination of TRSs. However, as those can be seen as a general framework for expressing computation, this opens up possibility for applications to *proving termination of programming languages*. So far, there has been work on applying term rewriting termination techniques to: functional programs [Gie95, GSSKT06] and logic programs [SKGST06, SKGST08].

In recent years, the emphasis in this field is on *automation of proving termination*. Many tools have been developed to fully automatically prove termination and an annual competition is organized where these tools compete on a large set of problems.

The complexity of these tools and the proofs produced by them makes ensuring their reliability a big challenge. Thus the need for *certification of termination proofs*, which encompasses formal verification of termination proofs using a theorem prover/checker.

Contributions

The contributions of this thesis, which we summarize below, can be divided into two categories: new methods and refinements to existing methods for proving termination of rewriting and contributions adding to the progress in the area of certification of termination proofs.

Part I	
Matrix-based termination methods (Chapter 2)	
Formalization of the monotone algebras framework and the matrix interpretation method [EWZ08].	Section 2.3.4.
Extension of the arctic interpretation method (matrix interpretations over the max/plus arctic semiring) for string rewriting [Wal07a] to term rewriting and generalization to arctic "below-zero" (integers allowed in place of natural numbers).	Section 2.2.
Formalization of the two aforementioned variants of arctic interpretations.	Section 2.3.5
Termination methods based on semantic labeling (Chapter 3)	
Automation of semantic labeling, in combination with RPO, for infinite models over natural numbers.	Section 3.1
SAT-based automation of predictive labeling over natural numbers within the dependency pairs setting.	Section 3.2
Automatic termination proving (Chapter 4)	
TPA (Termination Proved Automatically), first termination tool utilizing semantic labeling over infinite models; capable of producing certified termination proofs.	Chapter 4

Application to proving liveness properties by transformation to termination problems (Chapter 5)	
Generalization of the results in [GZ03a] to liveness with fairness.	Section 5.1
An alternative transformation (from liveness properties to termination problems) to the one from [GZ03a].	Section 5.2
Part II	
Higher-order rewriting (Chapters 6-9)	
Formalization of multisets and two variants of a multiset ordering.	Chapter 7
Formalization of the simply-typed lambda calculus.	Chapter 8
Formalization of the computability predicate proof method [Tai67].	Section 9.2
Formalization of HORPO — the higher-order variant of the recursive path ordering [JR99].	Section 9.3

Suggested Method of Reading

Parts I and II are completely independent and can be read on their own.

In Part I, Chapter 1 gives an introduction to term rewriting and to termination methods. It defines the general notions used in the following chapters and serves as a basis for this part of the thesis. Chapters 2 to 5 are independent from each other and can be read on their own, besides basic definitions that have to be recalled from Chapter 1.

In Part II, Chapter 6 provides a general introduction and a road-map to this part of the thesis. Chapters 7 to 9 build on each other and, unless the reader is familiar with some material, are best read sequentially.

Part I

First-Order Term Rewriting

Chapter 1

Term Rewriting Systems

In this chapter we will introduce the notions of term rewriting with an emphasis on termination. We will present some general notions concerning relations in Section 1.1. Then we will introduce term rewriting in Section 1.2 and termination of rewriting in Section 1.3. Finally in Section 1.4 we present the concept of certification of termination and introduce the CoLoR project.

1.1 Relations

Let $\mathcal{S}$ be an arbitrary set and $\rightarrow$ be an arbitrary binary relation over $\mathcal{S}$, *i.e.*, $\rightarrow \subseteq \mathcal{S} \times \mathcal{S}$. We write $\rightarrow^*$ for the reflexive transitive closure of $\rightarrow$ and $\rightarrow^+$ for its transitive closure. Define $<$ as the *converse* of $>$, that is $a < b \iff b > a$.

We introduce orders and a relation modulo.

Definition 1.1. We say that a relation $\geqslant$ is a *quasi-order* if it is reflexive and transitive.

A quasi-order $\geqslant$ that is additionally antisymmetric is a *partial order*.

A relation $>$ is a *strict partial order* if it is irreflexive and transitive (and hence antisymmetric). $\diamond$

Definition 1.2. Given two relations $\rightarrow$ and $\tilde{\rightarrow}$ we introduce a new relation $\rightarrow / \tilde{\rightarrow} \equiv \tilde{\rightarrow}^* \cdot \rightarrow$, which reads "$\rightarrow$ relative to $\tilde{\rightarrow}$". So $\rightarrow / \tilde{\rightarrow}$ denotes a number (possibly zero) of $\tilde{\rightarrow}$ steps followed by a single $\rightarrow$ step. $\diamond$

We continue with the introduction of normal forms and the concept of termination of a relation.

Definition 1.3. [Normal forms] We define the set of *normal forms* of $\rightarrow$ as $\mathrm{NF}(\rightarrow) \equiv \{s \in \mathcal{S} \mid \neg\exists_{s' \in \mathcal{S}}\ s \rightarrow s'\}$ and the set of normal forms reachable from a given set $\mathcal{I} \subseteq \mathcal{S}$ as $\mathrm{NF}(\mathcal{I}, \rightarrow) \equiv \{s \in \mathrm{NF} \mid \exists_{t \in \mathcal{I}}\ t \rightarrow^* s\}$. $\diamond$

Definition 1.4. [Termination] We denote *termination* of reduction sequences starting in $\mathcal{I}$ by $\mathrm{SN}(\mathcal{I}, \rightarrow)$, that is: $\mathrm{SN}(\mathcal{I}, \rightarrow) \equiv \neg\exists_{t_1, t_2, \ldots}\ t_1 \in \mathcal{I} \wedge \forall_i\ t_i \rightarrow t_{i+1}$. Termination of a relation, indicating lack of infinite reduction sequences, is denoted by $\mathrm{SN}(\rightarrow)$ and defined as $\mathrm{SN}(\rightarrow) \equiv \mathrm{SN}(\mathcal{S}, \rightarrow)$. Note that $\mathrm{SN}(\rightarrow)$ coincides with well-foundedness of $\rightarrow$. $\diamond$

We define a lexicographic extension that lifts a relation on elements to a relation on sequences of elements.

Definition 1.5. [Lexicographic extension] Let $>_1, \ldots, >_n$ be n relations on sets $A_1, \ldots, A_n$, for some fixed n. The *lexicographic extension* $(>_1, \ldots, >_n)_{lex}$ on tuples $A_1 \times \ldots \times A_n$ is defined as:

$(a_1, \ldots, a_n)\ (>_1, \ldots, >_n)_{lex}\ (a'_1, \ldots, a'_n)$ iff:

- $a_1 >_1 a'_1$, or
- $a_1 = a'_1$ and $(a_2, \ldots, a_n)\ (>_2, \ldots, >_n)_{lex}\ (a'_2, \ldots, a'_n)$ $\diamond$

An important property of the lexicographic extension is that it preserves well-foundedness.

Theorem 1.6. *Let $>_A$ and $>_B$ be two relations on sets A and B, respectively. If $>_A$ and $>_B$ are well-founded then $(>_A, >_B)_{lex}$ is well-founded.* $\square$

Now we will introduce the notion of well-founded induction.

Definition 1.7. [Accessibility] Let $\mathcal{W}$ be a set and $>$ a relation on $\mathcal{W}$. We define the *accessible part* of $\mathcal{W}$, denoted as $\mathcal{W}^{Acc}_<$, inductively as follows:

$$\frac{\forall_{y<x}\ y \in \mathcal{W}^{Acc}_<}{x \in \mathcal{W}^{Acc}_<} \qquad \diamond$$

Intuitively, $\mathcal{W}^{Acc}_<$ contains all elements of $\mathcal{W}$ that initiate only finite sequences. It easily follows that $<$ is well-founded if and only if $\mathcal{W}^{Acc}_< = \mathcal{W}$, so if every element of $\mathcal{W}$ is accessible.

The following well-founded part induction principle will be crucial for the proof of well-foundedness of multiset extensions in Section 7.3.

Definition 1.8. [Well-founded part induction] Let P be a predicate over $\mathcal{W}$. Then we define the following *well-founded part induction principle*:

$$\frac{\forall_{x \in \mathcal{W}^{Acc}_<}\ (\forall_{y<x}\ P(y)) \implies P(x)}{\forall_{x \in \mathcal{W}^{Acc}_<}\ P(x)} \qquad \diamond$$

1.2 Term Rewriting

In this section we briefly introduce term rewriting. For more details we refer to, for instance, [BN98, TeR03]. Here we restrict to first-order rewriting only; higher-order rewriting will be addressed in Part II of this thesis.

A *term rewriting system* (TRS) consists of rewrite rules formed with terms. For that we need to define terms, which essentially correspond to terms of first-order predicate logic. They are defined over a first-order signature.

Definition 1.9. [Signature] A *signature* $\mathcal{F}$ is a non-empty set of *function symbols*, equipped with an *arity function*, arity : $\mathcal{F} \to \mathbb{N}$, indicating how many arguments each function symbol expects. $\diamond$

Now we are ready to define terms.

Definition 1.10. [Terms] The set of *terms* over function symbols $\mathcal{F}$ and over an infinite set of variables $\mathcal{V}$, disjoint from $\mathcal{F}$, is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and defined inductively as:

- $x \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, for $x \in \mathcal{V}$,
- $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, for $f \in \mathcal{F}$, $\mathrm{arity}(f) = n$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.

The *subterm relation* on terms is denoted by $\trianglerighteq$ and defined as the smallest relation such that $t \trianglerighteq t$ and $f(x_1, \ldots, t_i, \ldots t_n) \trianglerighteq t_i$ for all $f \in \mathcal{F}$ and $1 \leqslant i \leqslant n = \mathrm{arity}(f)$. The *strict subterm relation* is denoted by $\triangleright$ and defined as: $t \triangleright u \iff t \trianglerighteq u \wedge t \neq u$.

For a term t we denote the set of function symbols occurring in it by $\mathcal{F}\mathrm{un}(t)$ and, similarly, the set of variables by $\mathcal{V}\mathrm{ar}(t)$. If $\mathcal{V}\mathrm{ar}(t) = \varnothing$ then t is *closed* (*ground*). If every variable occurs at most once in term t then t is called *linear*. The *root symbol* of a term t is denoted by $\mathrm{root}(t)$ and we have $\mathrm{root}(f(t_1, \ldots t_n)) = f$ and $\mathrm{root}(x) = x$. $\diamond$

Now let us define the notion of a rewrite rule.

Definition 1.11. [Rewrite rule] A *rewrite rule* is a pair of terms (l, r) with $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\mathcal{V}\mathrm{ar}(r) \subseteq \mathcal{V}\mathrm{ar}(l)$. A rewrite rule (l, r) is usually written as $l \to r$. $\diamond$

So rewrite rules are pairs of terms, such that the left-hand side is not a variable and every variable occurring in the right-hand side must also occur in the left-hand side of a rule. Term rewriting system is simply a set of rules.

Definition 1.12. [Term rewriting system, TRS] A *term rewriting system* (TRS) is a pair $\mathcal{R} = (\mathcal{F}, R)$ of a signature $\mathcal{F}$ and a set of rewrite rules R. The signature is usually left implicit and a TRS is identified with its set of rewrite rules.

Given a TRS $\mathcal{R}$ and a function symbol f we define

$$\mathcal{R}_f = \{l \to r \in \mathcal{R} \mid \text{root}(l) = f\}$$

that is $\mathcal{R}_f$ is the subset of $\mathcal{R}$ consisting of those rules that have f as the root of the left-hand side.

For a TRS $\mathcal{R}$ and a relation $\succ$ we define $\mathcal{R}_\succ = \{l \to r \in \mathcal{R} \mid l \succ r\}$. $\diamond$

An interesting sub-class of TRSs is that of string rewriting systems.

Definition 1.13. [String rewriting system, SRS] A TRS with all functions having arity one is called a *string rewriting system* (SRS). For SRSs it is customary to write terms as strings, hence $a_1(a_2(\ldots(a_n(x))\ldots))$ becomes $a_1a_2\cdots a_n$ and x becomes ϵ. $\diamond$

Now we introduce the rewrite relation associated with a given TRS. For that we need a notion of a substitution.

Definition 1.14. [Substitution] A *substitution* is a mapping from variables to terms $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$. An application of a substitution σ to a term t is written as $t\sigma$ and defined as follows:

$$\begin{aligned} x\sigma &= \sigma(x) \\ f(t_1, \ldots, t_n)\sigma &= f(t_1\sigma, \ldots, t_n\sigma) \end{aligned}$$ $\diamond$

Definition 1.15. [Rewrite relation] For a TRS $\mathcal{R}$ the *top rewrite relation* $\xrightarrow{\epsilon}_\mathcal{R}$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is defined by $t \xrightarrow{\epsilon}_\mathcal{R} u$ if and only if there is a rewrite rule $l \to r \in \mathcal{R}$ and a substitution $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $t = l\sigma$ and $u = r\sigma$.

The *rewrite relation* $\to_\mathcal{R}$ is a context closure of $\xrightarrow{\epsilon}_\mathcal{R}$; that is it is the smallest relation such that $\xrightarrow{\epsilon}_\mathcal{R} \subseteq \to_\mathcal{R}$ and if $t_i \to_\mathcal{R} t_i'$ then $f(t_1, \ldots, t_i, \ldots, t_n) \to_\mathcal{R} f(t_1, \ldots, t_i', \ldots, t_n)$ for every $f \in \mathcal{F}$ with $\text{arity}(f) = n$. If $t \to_\mathcal{R} u$ then we say that there is a rewrite step from t to u or, simply, that t rewrites to u.

Note that we will often write $\to$ instead of $\to_\mathcal{R}$ if the TRS $\mathcal{R}$ is clear from the context. $\diamond$

Example 1.16. *Let us present a very simple example of a TRS for addition of natural numbers. Natural numbers are represented with a constant* 0 *and an unary function symbol* s *for successor. We will define addition represented by a binary* plus *operator. This gives us signature:*

$$\mathcal{F} = \{0, \text{s}, \text{plus}\}$$

the arity function:

$$\text{arity}(0) = 0 \qquad \text{arity}(\text{s}) = 1 \qquad \text{arity}(\text{plus}) = 2$$

and the following two rewrite rules:

$$\mathrm{plus}(0, y) \to y$$
$$\mathrm{plus}(\mathrm{s}(x), y) \to \mathrm{s}(\mathrm{plus}(x, y))$$

We also have the following sequence of rewrite steps:

$$\mathrm{plus}(\mathrm{s}(\mathrm{s}(0)), \mathrm{s}(\mathrm{s}(0))) \to \mathrm{s}(\mathrm{plus}(\mathrm{s}(0), \mathrm{s}(\mathrm{s}(0)))) \to \mathrm{s}(\mathrm{s}(\mathrm{plus}(0, \mathrm{s}(\mathrm{s}(0))))) \to \mathrm{s}(\mathrm{s}(\mathrm{s}(\mathrm{s}(0))))$$ $\lhd$

1.3 Termination of Term Rewriting

Termination is an important concept in term rewriting. Many methods for proving termination have been proposed over the years, some of which we will introduce in this section.

Recently the emphasis in this area is on automation and a number of tools have been developed for that purpose. One of such tools, TPA [Kop06c], will be presented in Chapter 4. Most of the termination methods require a number of parameters, giving rise to a (often large or even infinite) search space and the tools aim at efficiently exploring (a finite part) of it. A recent trend in the field is to use satisfiability solving (SAT) for that purpose. A proof search problem is then encoded as a propositional formula in such a way that a satisfying assignment can be translated back to the parameters required to successfully apply a given method.

To evaluate termination tools and stimulate their improvement the annual Termination Competition [TC] is organized, where such tools compete on a set of problems from the Termination Problems Database (TPDB), [TPD]. This competition has become a de-facto standard in evaluation of new termination techniques and developments of termination tools.

All performance measures presented in this thesis will be performed on the TPDB. Whenever possible, the selection of problems from version 4.2 of this database will be used, consisting of 975 TRSs and 517 SRSs. This is the same selection as used in the 2007 edition of the termination competition.

1.3.1 Definitions

We begin with definitions of termination and relative termination.

Definition 1.17. [(Relative) termination] A TRS $\mathcal{R}$ is called *terminating* iff there is no infinite reduction $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots$, *i.e.*, $\mathrm{SN}(\to_{\mathcal{R}})$.

A TRS $\mathcal{R}$ is called *terminating relative to* a TRS $\mathcal{S}$ iff there is no infinite reduction $t_1 \to_{\mathcal{R}\cup\mathcal{S}} t_2 \to_{\mathcal{R}\cup\mathcal{S}} \cdots$ containing infinitely many $\to_{\mathcal{R}}$ steps. Note that this coincides with $\mathrm{SN}(\to_{\mathcal{R}} / \to_{\mathcal{S}})$.

We will write $\mathrm{SN}(\mathcal{R})$ instead of $\mathrm{SN}(\to_{\mathcal{R}})$ and $\mathrm{SN}(\mathcal{R}/\mathcal{S})$ instead of $\mathrm{SN}(\to_{\mathcal{R}} / \to_{\mathcal{S}})$. We will also use $\mathcal{R}_{\text{top}}$ interchangeably with $\xrightarrow{\epsilon}_{\mathcal{R}}$.

We write $\mathcal{SN}$ for the subset of $\mathcal{T}(\mathcal{F},\mathcal{V})$ consisting of all terminating terms and $\mathcal{T}_\infty$ for the set of minimal non-terminating terms, *i.e.*, non-terminating terms all of whose arguments are terminating. $\diamond$

Now we introduce the notion of $\mathcal{F}$-algebras, that will play an important role in interpretation-based methods for proving termination.

Definition 1.18. [$\mathcal{F}$-algebra] A *$\mathcal{F}$-algebra*, $(A, \{f_A\}_{f\in\mathcal{F}})$ consists of a non-empty set A together with a map $[f_A] : A^n \to A$ for every $f \in \mathcal{F}$, where $\mathrm{arity}(f) = n$. $\diamond$

Definition 1.19. Given a $\mathcal{F}$-algebra, $(A, \{f_A\}_{f\in\mathcal{F}})$, we define the interpretation of terms $[\cdot]_\alpha$ for a given evaluation of variables $\alpha : \mathcal{V} \to A$, as:

$$\begin{aligned} [x]_\alpha &= \alpha(x) \quad \text{for } x \in \mathcal{V} \\ [f(t_1,\ldots,t_n)]_\alpha &= [f_A]([t_1]_\alpha,\ldots,[t_n]_\alpha). \end{aligned}$$ $\diamond$

Now we introduce the notion of reduction orderings and a theorem employing them for proving termination.

Definition 1.20. [Reduction ordering] A *Reduction ordering* is a relation $>$ on terms that is:

- well-founded,
- closed under substitutions, *i.e.*, if $t > u$ then $t\sigma > u\sigma$ for every substitution σ and
- closed under contexts, *i.e.*, if $t > u$ then for all $s_1,\ldots,s_n$ and all f with $\mathrm{arity}(f) = n$, $f(s_1,\ldots,t,\ldots,s_n) > f(s_1,\ldots,u,\ldots,s_n)$. $\diamond$

Theorem 1.21 ([Lan79])**.** *A TRS $\mathcal{R}$ is terminating,* $\mathrm{SN}(\mathcal{R})$*, iff there exists a reduction order $>$ such that $\mathcal{R} \subseteq >$.* $\square$

The notion of reduction pairs extends that of a reduction order and will be used in Section 1.3.7.

Definition 1.22. [Reduction pairs] A *reduction pair* $(\gtrsim, >)$ consists of a quasi-order $\gtrsim$ which is closed under contexts and substitutions and a well-founded order $>$ which is closed under substitutions such that the inclusion $\gtrsim \cdot > \subseteq >$ or the inclusion $> \cdot \gtrsim \subseteq >$ holds.

We say that a reduction pair $(\gtrsim, >)$ is *$\mathcal{C}_{\mathcal{E}}$-compatible* iff $\mathrm{c}(x,y) \gtrsim x$ and $\mathrm{c}(x,y) \gtrsim y$, where c is a new binary function symbol. $\diamond$

Now we shortly introduce the notion of simple termination.

Definition 1.23. [Simple termination] A *simplification order* is a reduction order $>$, that has the subterm property, *i.e.*, $\rhd \subseteq >$.

A TRS $\mathcal{R}$ is called *simply terminating* iff there exists a simplification order $>$ such that $\mathcal{R} \subseteq >$. $\diamond$

1.3.2 Termination with Weakly Monotone Algebras

Here we summarize the monotone algebra theory as presented in [EWZ08]. There is one difference: in contrast to [EWZ08] we do not consider many-sortedness. The reason for the more complex setup in [EWZ08] is that it allows for an optimization in the search for termination proofs using matrix interpretations. However, our prime focus for monotone algebras will be their use for certification of termination (see Section 1.4 for introduction to certification) and not the implementation details. Since every proof in the many-sorted setting can be trivially translated to the one-sorted setting, we opt for this simpler setup.

Definition 1.24. [Monotonicity] Let A be a non-empty set. A function $f : A^n \to A$ is *monotone* with respect to a binary relation $\to$ on A iff:

$$\forall_{a_1,\ldots,a_i,a_i',\ldots a_n \in A}\ a_i \to a_i' \implies f(a_1,\ldots,a_i,\ldots,a_n) \ \to\ f(a_1,\ldots,a_i',\ldots,a_n) \quad \diamond$$

Definition 1.25. [Weakly monotone $\mathcal{F}$-algebra] Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$. A well-founded *weakly monotone $\mathcal{F}$-algebra* is a quadruple $\mathcal{A} = (A, \{f_A\}_{f\in\mathcal{F}}, >, \gtrsim)$ such that:

- $(A, \{f_A\}_{f\in\mathcal{F}})$ is a $\mathcal{F}$-algebra,
- all algebra operations $[f]$ for every $f \in \mathcal{F}$ are weakly monotone, *i.e.*, monotone with respect to $\gtrsim$,
- $>$ is a well-founded relation on A, and
- relations $\gtrsim$ and $>$ are compatible, that is: $> \cdot \gtrsim \subseteq >$ or $\gtrsim \cdot > \subseteq >$.

An *extended monotone $\mathcal{F}$-algebra* $(A, \{f_A\}_{f\in\mathcal{F}}, >, \gtrsim)$ is a weakly monotone $\mathcal{F}$-algebra $(A, \{f_A\}_{f\in\mathcal{F}}, >, \gtrsim)$ in which moreover for every $f \in \mathcal{F}$ the operation $[f]$ is strictly monotone, *i.e.*, monotone with respect to $>$. $\diamond$

Definition 1.26. For a weakly monotone $\mathcal{F}$-algebra $\mathcal{A} = (A, \{f_A\}_{f\in\mathcal{F}}, >, \gtrsim)$ we extend the order $\gtrsim$ on A to an order $\gtrsim_{\mathcal{A}}$ on terms, as

$$t \gtrsim_{\mathcal{A}} u \iff \forall_{\alpha:\mathcal{V}\to A}\ [t]_\alpha \gtrsim [u]_\alpha$$

$>$ is extended to $>_{\mathcal{A}}$ in a similar way. $\diamond$

Up to presentation details the following theorem is the one-sorted version of the main monotone algebras theorem from [EWZ08, Theorem 2].

Theorem 1.27. *Let $\mathcal{R}, \mathcal{R}', \mathcal{S}, \mathcal{S}'$ be TRSs over a signature $\mathcal{F}$.*

(a) Let $(A, \{f_A\}_{f \in \mathcal{F}}, >, \gtrsim)$ be an extended monotone $\mathcal{F}$-algebra such that $l \gtrsim_{\mathcal{A}} r$ for every rule $l \to r$ in $\mathcal{R} \cup \mathcal{S}$ and $l >_{\mathcal{A}} r$ for every rule $l \to r$ in $\mathcal{R}' \cup \mathcal{S}'$.

Then $\mathrm{SN}(\mathcal{R}/\mathcal{S})$ *implies* $\mathrm{SN}(\mathcal{R} \cup \mathcal{R}'/\mathcal{S} \cup \mathcal{S}')$.

(b) Let $(A, \{f_A\}_{f \in \mathcal{F}}, >, \gtrsim)$ be a weakly monotone $\mathcal{F}$-algebra such that $l \gtrsim_{\mathcal{A}} r$ for every rule $l \to r$ in $\mathcal{R} \cup \mathcal{S}$ and $l >_{\mathcal{A}} r$ for every rule $l \to r$ in $\mathcal{R}'$.

Then $\mathrm{SN}(\mathcal{R}_{\mathrm{top}}/\mathcal{S})$ *implies* $\mathrm{SN}(\mathcal{R}_{\mathrm{top}} \cup \mathcal{R}'_{\mathrm{top}}/\mathcal{S})$. □

1.3.3 Polynomial Interpretations

Polynomial interpretation method [Lan79] is a well-known termination technique, dating back to the late 70s. We will use it as an illustration for the monotone algebra setting from the previous section.

To prove (relative) termination with Theorem 1.27 we need to:

1. provide a *domain* for the algebra, A,
2. provide *two orders*, $>, \gtrsim$, over the domain A,
3. check that the *orders are compatible*, *i.e.*, $> \cdot \gtrsim \subseteq >$ or $\gtrsim \cdot > \subseteq >$ and $>$ *is well-founded*,
4. fix a *class of functions used for interpretation* of symbols, $[f_A] : A^n \to A$, for symbol f of arity n,
5. prove that this class of functions is *weakly monotone*, *i.e.*, monotone with respect to the chosen order $\gtrsim$,
6. if proving termination by means of Theorem 1.27a is intended then prove that it is also *strictly monotone*, *i.e.*, monotone with respect to $>$,
7. finally for application of Theorem 1.27, we need to be able to *compare interpretations* of terms with respect to $>_{\mathcal{A}}$ and $\gtrsim_{\mathcal{A}}$, *i.e.*, provide properties ensuring that one interpretation is always greater (greater equal) than some other interpretation, regardless of interpretations of variables.

We will now illustrate all those steps on the polynomial interpretation method. We will wrap up this section with an example showing this method in practice.

For polynomial interpretations we:

1. Fix the domain to natural numbers, $\mathbb{N}$,
2. with the standard orders $>_{\mathbb{N}}, \geqslant_{\mathbb{N}}$.
3. Compatibility of those orders is immediate and so is well-foundedness of $>_{\mathbb{N}}$.
4. An interpretation of an n-ary function symbol f is a polynomial over a vector of variables $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{N}^n$ with natural numbers as coefficients, $\mathcal{P}_{\mathbb{N}}(\vec{x})$.
5. Such polynomials are clearly weakly monotone (coefficients are natural numbers).
6. To obtain strict monotonicity we must require that every variable occurs positively. So for a polynomial $\mathcal{P}_{\mathbb{N}}(\vec{x})$ over a vector of variables $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{N}^n$ we require that it contains a monomial $c * x_i$, with $c > 0$, for $1 \leqslant i \leqslant n$.
7. Comparison of polynomials amounts to the positiveness check. Let $\vec{x} = (x_1, \ldots, x_n)$. Suppose we want to compare polynomials $\mathcal{P}_{\mathbb{N}}(\vec{x})$ and $\mathcal{Q}_{\mathbb{N}}(\vec{x})$. We can conclude

   $$\forall_{\vec{x} \in \mathbb{N}^n}\ \mathcal{P}_{\mathbb{N}}(\vec{x}) \geqslant \mathcal{Q}_{\mathbb{N}}(\vec{x})$$

   iff:

   $$\forall_{\vec{x} \in \mathbb{N}^n}\ \mathcal{P}_{\mathbb{N}}(\vec{x}) - \mathcal{Q}_{\mathbb{N}}(\vec{x}) \geqslant 0$$

   To compare polynomials strictly we simply consider the polynomial

   $$\mathcal{P}_{\mathbb{N}}(\vec{x}) - \mathcal{Q}_{\mathbb{N}}(\vec{x}) - 1$$

   in the above inequality. Note that we need to allow integer coefficients to properly define the difference operation on polynomials.

   This problem is undecidable in general. There are however heuristics providing conditions under which we can conclude that such inequality holds. The method of choice is usually the absolute positiveness check by the shifting method of [HJ98]. It states that evaluation of a polynomial is always greater (resp. greater or equal) than 0 if all of its coefficients are non-negative and the constant factor is positive (resp. non-negative).

   When comparing two polynomials this amounts to the fact that the evaluation of one polynomial is greater or equal than the evaluation of some other polynomial if all the coefficients of the first polynomial are greater or equal than the respective coefficients in the second one. It is strictly greater if, additionally, the constant factor is strictly greater.

Let us illustrate this method on an example.

Example 1.28. *Consider the TRS from Example 1.16 and the following polynomial interpretation:*

$$[\mathrm{plus}(x,y)] = 2x + y \qquad [s(x)] = x + 1 \qquad [0] = 0$$

It gives us the following interpretations for the rules:

$$\begin{aligned}[\mathrm{plus}(0,y)] &= \quad y = y \quad = [y]\\ [\mathrm{plus}(\mathrm{s}(x),y)] &= 2x + y + 2 > 2x + y + 1 = [\mathrm{s}(\mathrm{plus}(x,y))]\end{aligned}$$

The interpretations for the first rule are equal and we get a strict decrease for the second rule — all coefficients are equal except for the constant which is bigger for the left-hand side — so, according to Theorem 1.27 we can remove this rule. Then the termination of the first rule alone is easy (for instance take polynomial interpretation with $[\mathrm{plus}(x,y)] = x + y$ *and* $[0] = 1$. $\lhd$

1.3.4 Recursive Path Order

Recursive path ordering (RPO) is an ordering introduced by Dershowitz [Der82]. It extends a well-founded order on the signature, called a *precedence*, to a reduction order on terms.

Definition 1.29. [RPO] Let $\mathcal{F}$ be any signature, possibly infinite. Let $\succ$ be a well-founded partial order on $\mathcal{F}$ (strict or non-strict), called a *precedence*. Let τ be a *status function* on $\mathcal{F}$, *i.e.*, $\tau : \mathcal{F} \to \{\mathrm{Lex}, \mathrm{Mul}\}$. The *recursive path order* (RPO) on terms, $\succ^{\mathsf{RPO}}$, is defined as: $s = f(s_1, \ldots, s_n) \succ^{\mathsf{RPO}} t$ iff:

- $s_i = t$ or $s_i \succ^{\mathsf{RPO}} t$ for some $i \in \{1, \ldots, n\}$, or
- $t = g(t_1, \ldots, t_m)$, $s \succ^{\mathsf{RPO}} t_i$ for all $i \in \{1, \ldots, m\}$, and either
  - $f \succ g$, or
  - $f = g$ and $\langle s_1, \ldots, s_n\rangle \succ^{\mathsf{RPO}}_{\tau(f)} \langle t_1, \ldots, t_m\rangle$.

The $\succ^{\mathsf{RPO}}_{\tau(f)}$ in the second case depends on the status of f. If $\tau(f) = Lex$ then this becomes the lexicographic extension of $\succ^{\mathsf{RPO}}$ — $\succ^{\mathsf{RPO}}_{lex}$ as in Definition 1.5. If $\tau(f) = Mul$ then the multiset extension $\succ^{\mathsf{RPO}}_{mul}$ is used (see Definition 7.3 for the multiset extension of a relation).

The lexicographic path order, LPO (resp. multiset path order, MPO) is defined similarly to RPO only no status is given and $\tau(f) = \mathrm{Lex}$ (resp. $\tau(f) = \mathrm{Mul}$) for every $f \in \mathcal{F}$. $\diamond$

Note that in Definition 1.5 we presented the lexicographic with the left-to-right comparison of elements. A straightforward extension allows to permute elements before comparing them, which requires extension of the RPO status, in the *Lex* case, with such a permutation of function arguments. Also note that RPO is a simplification order.

The main property of this order is the following:

Theorem 1.30 ([Der82])**.** *Let $\mathcal{R}$ be a TRS over $\mathcal{F}$ and let $\succ$ be a well-founded precedence and τ be a status function for $\mathcal{F}$. If $\mathcal{R} \subseteq \succ^{\mathsf{RPO}}$, then* $\mathrm{SN}(\mathcal{R})$. □

Let us illustrate an application of this theorem on an example.

Example 1.31. *Let us consider the TRS from Example 1.16. Consider a precedence with* $\mathrm{plus} \succ \mathrm{s}$ *and the status function* $\tau(f) = \mathrm{Lex}$ *for all* $f \in \mathcal{F}$. *Then both rules of this TRS can be oriented with RPO hence proving termination of this system.* ◁

So for proving termination by RPO one has to find a well-founded precedence $\succ$ and a status function τ such that $l \succ^{\mathsf{RPO}} r$ for every rule $l \to r$. In tools this is typically done by collecting constraints on $\succ$ and checking whether these constraints give rise to a well-founded precedence $\succ$. In this process usually many choices are possible, both for the precedence and status function, which are typically handled with back-tracking. The crucial algorithm required in this procedure is to check whether a set of constraints on $\succ$ gives rise to a well-founded precedence $\succ$. For finite signatures this coincides with checking whether the corresponding graph is acyclic.

1.3.5 Semantic Labeling

Semantic labeling is a transformational termination technique for TRSs due to Zantema [Zan95, Zan03]. The core idea is to interpret the function symbols in some model and use this interpretation to label the system. After applying the transformation some function symbols are equipped with a label and the resulting TRS is terminating if and only if the original one is terminating. This approach is often successful in the sense that proving termination for the labeled system is easier than for the original one. Non-simply terminating systems are often transformed to systems for which termination is easily proved by polynomials or RPO – we will present one example of that by the end of this section.

First we recall the main theory of semantic labeling, presenting only its quasi-model variant.

Definition 1.32. [Quasi-model] Let $\mathcal{R}$ be a TRS. A weakly monotone $\mathcal{F}$-algebra $(A, \{f_A\}_{f \in \mathcal{F}}, >, \gtrsim)$ is called a *quasi-model* for $\mathcal{R}$ iff $\mathcal{R} \subseteq \gtrsim_{\mathcal{A}}$. ⋄

Now we introduce the labeling, labeled signature and a function used to label a TRS.

Definition 1.33. [Labeling] Let $\mathcal{R}$ be a TRS over $\mathcal{F}$ and let $(A, \{f_A\}_{f\in\mathcal{F}}, >, \gtrsim)$ be a quasi-model for $\mathcal{R}$. A weakly monotone *labeling* $\mathcal{L}$ consists of a set of labels $L_f \subseteq A$ together with a mapping $\ell_f : A^n \to L_f$ for every function symbol $f \in \mathcal{F}$ with arity$(f) = n$, such that ℓ_f is weakly monotone in all coordinates. ◇

Definition 1.34. [Labeled signature, $\mathcal{F}_{\mathsf{lab}}$] Let $\mathcal{R}$ be a TRS over $\mathcal{F}$ and let $\mathcal{L}$ be a labeling for $\mathcal{R}$. The labeled signature $\mathcal{F}_{\mathsf{lab}}$ consist of n-ary function symbols f_a for every n-ary function symbol $f \in \mathcal{F}$ and every label $a \in L_f$ together with all function symbols $f \in \mathcal{F}$ such that $L_f = \varnothing$. ◇

Definition 1.35. [lab_α] Let $\mathcal{R}$ be a TRS over $\mathcal{F}$ and $\mathcal{L}$ be a labeling for $\mathcal{R}$. We extend an assignment of variables $\alpha\colon \mathcal{V} \to A$ to the mapping $\mathsf{lab}_\alpha : \mathcal{T}(\mathcal{F},\mathcal{V}) \to \mathcal{T}(\mathcal{F}_{\mathsf{lab}},\mathcal{V})$ in the following way:

$$\mathsf{lab}_\alpha(t) = \begin{cases} t & \text{if } t \text{ is a variable,} \\ f(\mathsf{lab}_\alpha(t_1),\dots,\mathsf{lab}_\alpha(t_n)) & \text{if } t = f(t_1,\dots,t_n) \text{ and } L_f = \varnothing, \\ f_a(\mathsf{lab}_\alpha(t_1),\dots,\mathsf{lab}_\alpha(t_n)) & \text{if } t = f(t_1,\dots,t_n) \text{ and } L_f \neq \varnothing, \end{cases}$$

where a denotes the label $\ell_f([t_1]_\alpha,\dots,[t_n]_\alpha)$. ◇

Finally we define the labeled system. But first we present the definition of the so-called decreasing rules.

Definition 1.36. [$\mathcal{D}\mathrm{ec}_\mathcal{R}$] Let $\mathcal{R}$ be a TRS over $\mathcal{F}$ and $\mathcal{L}$ be a labeling for $\mathcal{R}$. We define the TRS $\mathcal{D}\mathrm{ec}_\mathcal{R}$ to consist of all rewrite rules

$$f_a(x_1,\dots,x_n) \to f_b(x_1,\dots,x_n)$$

with $f \in \mathcal{F}$, arity$(f) = n$, $a, b \in L_f$ with $a > b$, and pairwise different variables $x_1,\dots,x_n$. ◇

Definition 1.37. [Labeled TRS, $\mathcal{R}_{\mathsf{lab}}$] Let $\mathcal{R}$ be a TRS, $(A, \{f_A\}_{f\in\mathcal{F}}, >, \gtrsim)$ a quasi-model for $\mathcal{R}$ and $\mathcal{L}$ a labeling for $\mathcal{R}$. We define the labeled TRS $\mathcal{R}_{\mathsf{lab}}$ over the signature $\mathcal{F}_{\mathsf{lab}}$ to consist of the rules:

$$\mathsf{lab}_\alpha(l) \to \mathsf{lab}_\alpha(r)$$

for all $\alpha : \mathcal{V} \to A$ and all rules $l \to r \in \mathcal{R}$. ◇

Now the main theorem of semantic labeling reads:

Theorem 1.38 ([Zan95])**.** *Let $\mathcal{R}$ be a TRS over $\mathcal{F}$, $(A, \{f_A\}_{f\in\mathcal{F}}, >, \gtrsim)$ be a quasi-model for $\mathcal{R}$ and $\mathcal{L}$ a labeling for $\mathcal{R}$. Then* SN$(\mathcal{R})$ *iff* SN$(\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}_\mathcal{R})$. □

Let us apply this theorem on an example.

Example 1.39. *Consider the SRS $\mathcal{R}$ consisting of the single rule:*

$$\mathrm{a\,a} \to \mathrm{a\,b\,a}$$

This TRS is non-simply terminating. Suppose it was. Then there would be a simplification order $>$ such that $\mathrm{a\,a} > \mathrm{a\,b\,a}$ but then $\mathrm{b} > \epsilon$ as $\mathrm{b} \trianglerighteq \epsilon$ (note that $\epsilon \equiv x$ and $\mathrm{b} \equiv \mathrm{b}(x)$ in the SRS notation) and $>$ is a simplification order. Then $>$ would be a witness for termination of $\mathcal{R} \cup \{\mathrm{b} \to \epsilon\}$, which is non-terminating:

$$\mathrm{a\,a} \to \mathrm{a\,b\,a} \to \mathrm{a\,a} \to \dots$$

This means that simplification orders, like RPO *introduced in Section 1.3.4, will fail to prove termination of this system. However it is easy to see that $\mathcal{R}$ is terminating. We will show how termination can be proven with semantic labeling.*

For the monotone algebra choose $A = \{0, 1\}$ with $1 > 0$. Choose the interpretations

$$[\mathrm{a}] = 1 \qquad [\mathrm{b}] = 0$$

both constant and hence weakly monotone. This interpretation is a quasi-model for $\mathcal{R}$ as

$$[\mathrm{a\,a}]_\alpha = 1 = [\mathrm{a\,b\,a}]_\alpha$$

for arbitrary α. For the labeling choose:

$$L_{\mathrm{a}} = A \qquad L_{\mathrm{b}} = \varnothing$$

and

$$\ell_{\mathrm{a}}(x) = x$$

We get the labeled signature

$$\mathcal{F}_{\mathsf{lab}} = \{\mathrm{a}_0, \mathrm{a}_1, \mathrm{b}\}$$

and the decreasing SRS

$$\mathcal{D}\mathrm{ec}_{\mathcal{R}} = \{\mathrm{a}_1 \to \mathrm{a}_0\}$$

The labeled SRS $\mathcal{R}_{\mathsf{lab}}$ consists of two rules:

$$\mathrm{a}_1\,\mathrm{a}_0 \to \mathrm{a}_0\,\mathrm{b}\,\mathrm{a}_0$$
$$\mathrm{a}_1\,\mathrm{a}_1 \to \mathrm{a}_0\,\mathrm{b}\,\mathrm{a}_1$$

Termination of $\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}_{\mathcal{R}}$ can be proven with RPO *with the precedence $\mathrm{a}_1 \succ \mathrm{a}_0 \succ \mathrm{b}$, hence, by application of Theorem 1.38, proving termination of $\mathcal{R}$.* $\lhd$

1.3.6 Predictive Labeling

Predictive labeling [HM06] is a variant of semantic labeling where the quasi-model condition is only demanded for a part of the system, namely for the so-called usable rules, induced by the labeling.

Definition 1.40. Let $\mathcal{R}$ be a TRS. For function symbols f and g we write $f \rhd_{\mathrm{d}} g$ if there exists a rewrite rule $l \to r \in \mathcal{R}$ such that $f = \mathrm{root}(l)$ and g is a function symbol in $\mathcal{F}\mathrm{un}(r)$.

For a set $F \subseteq \mathcal{F}$ we write $F^{\rhd_{\mathrm{d}}^*}$ for the set $\{g \mid f \rhd_{\mathrm{d}}^* g \text{ for some } f \in F\}$. ◇

Definition 1.41. Let $\mathcal{R}$ be a TRS, ℓ be a labeling for $\mathcal{R}$ and t a term. We define:

$$\mathcal{G}_\ell(t) = \begin{cases} \varnothing & \text{if } t \text{ is a variable,} \\ \mathcal{F}\mathrm{un}(t_1)^{\rhd_{\mathrm{d}}^*} \cup \cdots \cup \mathcal{F}\mathrm{un}(t_n)^{\rhd_{\mathrm{d}}^*} & \text{if } t = f(t_1, \ldots, t_n) \text{ and } L_f \neq \varnothing, \\ \mathcal{G}_\ell(t_1) \cup \cdots \cup \mathcal{G}_\ell(t_n) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } L_f = \varnothing. \end{cases}$$

Furthermore we define:

$$\mathcal{G}_\ell(\mathcal{R}) = \bigcup_{l \to r \in \mathcal{R}} \mathcal{G}_\ell(l) \cup \mathcal{G}_\ell(r)$$

and the set of *usable rules* for a labeling ℓ is defined as

$$\mathcal{U}(\ell) = \{l \to r \in \mathcal{R} \mid \mathrm{root}(l) \in \mathcal{G}_\ell(\mathcal{R})\}$$ ◇

The above definition was improved in [TM07] by showing that right-hand sides of the rules can be ignored in the computation of $\mathcal{G}_\ell(\mathcal{R})$. For the purpose of this thesis we will however stick to the above definition without this improvement, as presented in [HM06, KM07].

Typically, $\mathcal{U}(\ell)$ is a proper subset of $\mathcal{R}$. The point of predictive labeling is to replace the quasi-model condition $\mathcal{R} \subseteq \gtrsim_{\mathcal{A}}$ of semantic labeling by the easier to satisfy condition $\mathcal{U}(\ell) \subseteq \gtrsim_{\mathcal{A}}$. We illustrate this on an example.

Example 1.42. *Consider the following TRS from [TG05] (AProVE/rta1.trs in TPDB [TPD]):*

(1)	$\mathrm{plus}(0, y) \to y$	(5)	$\mathrm{plus}(\mathrm{s}(\mathrm{s}(x)), y) \to \mathrm{s}(\mathrm{plus}(x, \mathrm{s}(y)))$
(2)	$\mathrm{plus}(\mathrm{s}(0), y) \to \mathrm{s}(y)$	(6)	$\mathrm{plus}(x, \mathrm{s}(\mathrm{s}(y))) \to \mathrm{s}(\mathrm{plus}(\mathrm{s}(x), y))$
(3)	$\mathrm{ack}(0, y) \to \mathrm{s}(y)$	(7)	$\mathrm{ack}(\mathrm{s}(x), \mathrm{s}(y)) \to \mathrm{ack}(x, \mathrm{plus}(y, \mathrm{ack}(\mathrm{s}(x), y)))$
(4)	$\mathrm{ack}(\mathrm{s}(x), 0) \to \mathrm{ack}(x, \mathrm{s}(0))$		

Now suppose

$$L_{\mathrm{s}} \neq \varnothing \qquad L_0 = L_{\mathrm{plus}} = L_{\mathrm{ack}} = \varnothing.$$

Then applying the above definition gives

$$\mathcal{G}_\ell(\mathcal{R}) = \{\text{plus}, \text{s}\} \qquad \mathcal{U}(\ell) = \{1, 2, 5, 6\}.$$

This means that for predictive labeling rules $3, 4$ *and* 7 *can be ignored for checking the quasi-model constraints.* $\lhd$

The weakly monotone algebras $(A, \{f_A\}_{f \in \mathcal{F}}, >, \gtrsim)$ used to determine the labeling and to satisfy the quasi-model constraints in connection with predictive labeling must satisfy one additional property.

Definition 1.43. [Weakly monotone $\sqcup$-algebras] Let $(A, \{f_A\}_{f \in \mathcal{F}}, >, \gtrsim)$ be a weakly monotone algebra. We call it a weakly monotone $\sqcup$-algebra if for every finite subset $X \subseteq A$ there exists a least upper bound $\bigsqcup X$ of X in A (with respect to $>$). $\diamond$

The main theorem of predictive labeling can now be stated.

Theorem 1.44 ([HM06])**.** *Let $\mathcal{R}$ be a finitely branching TRS,* $(A, \{f_A\}_{f \in \mathcal{F}}, >, \gtrsim)$ *a weakly monotone $\sqcup$-algebra, and ℓ a weakly monotone labeling for $\mathcal{R}$ such that $\mathcal{U}(\ell) \subseteq \gtrsim_{\mathcal{A}}$. If $\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}$ is terminating then $\mathcal{R}$ is terminating.* $\square$

Due to the restriction to $\sqcup$-algebras, predictive labeling is less powerful than semantic labeling in theory. However, since the algebras used in current termination tools are $\sqcup$-algebras, in practice predictive labeling is to be preferred as it has the clear advantage of weakening the quasi-model condition; instead of all rules only the usable rules need to be oriented, which brings improvements in proving power as well as efficiency.

1.3.7 Dependency Pairs

The dependency pair method [AG00] is a powerful approach for proving termination of TRSs. The dependency pair framework [GTSK04] is a modular reformulation and improvement of this approach. We present a simplified version which is sufficient for our purposes. For further information on dependency pairs and more detailed explanation of the concepts introduced below the reader is referred to [AG00, GTSK04, GTSKF06, HM07].

Definition 1.45. [Dependency pairs] Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$. The set of *defined* symbols is defined as $\mathcal{D}_\mathcal{R} = \{\text{root}(l) \mid l \to r \in \mathcal{R}\}$. We extend a signature $\mathcal{F}$ to the signature $\mathcal{F}^\sharp$ by adding symbols $f^\sharp$ for every symbol $f \in \mathcal{D}_\mathcal{R}$. If $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with $\text{root}(t) \in \mathcal{D}_\mathcal{R}$ then $t^\sharp$ denotes the term that is obtained from t by replacing its root symbol with $\text{root}(t)^\sharp$.

If $l \to r \in \mathcal{R}$ and $t \trianglelefteq u$ with $\text{root}(t) \in \mathcal{D}_\mathcal{R}$ then the rule $l^\sharp \to t^\sharp$ is a *dependency pair* of $\mathcal{R}$. The set of all dependency pairs of $\mathcal{R}$ is denoted by $\text{DP}(\mathcal{R})$. $\diamond$

We illustrate this definition on an example:

Example 1.46. *For the TRS presented in Example 1.42 there are six dependency pairs:*

$$\begin{aligned}
&(8) && \mathrm{ack}^\sharp(\mathrm{s}(x), 0) \to \mathrm{ack}^\sharp(x, \mathrm{s}(0)) \\
&(9) && \mathrm{plus}^\sharp(\mathrm{s}(\mathrm{s}(x)), y) \to \mathrm{plus}^\sharp(x, \mathrm{s}(y)) \\
&(10) && \mathrm{plus}^\sharp(x, \mathrm{s}(\mathrm{s}(y))) \to \mathrm{plus}^\sharp(\mathrm{s}(x), y) \\
&(11) && \mathrm{ack}^\sharp(\mathrm{s}(x), \mathrm{s}(y)) \to \mathrm{ack}^\sharp(x, \mathrm{plus}(y, \mathrm{ack}(\mathrm{s}(x), y))) \\
&(12) && \mathrm{ack}^\sharp(\mathrm{s}(x), \mathrm{s}(y)) \to \mathrm{plus}^\sharp(y, \mathrm{ack}(\mathrm{s}(x), y)) \\
&(13) && \mathrm{ack}^\sharp(\mathrm{s}(x), \mathrm{s}(y)) \to \mathrm{ack}^\sharp(\mathrm{s}(x), y)
\end{aligned}$$

$\lhd$

Definition 1.47. [DP problems] A *DP problem* is a pair of TRSs $(\mathcal{P}, \mathcal{R})$. The problem is said to be *finite* if there is no infinite sequence

$$t_1 \to_{\mathcal{R}}^{*} s_1 \xrightarrow{\epsilon}_{\mathcal{P}} t_2 \to_{\mathcal{R}}^{*} s_2 \xrightarrow{\epsilon}_{\mathcal{P}} \cdots$$

such that all terms $t_1, t_2, \ldots$ are terminating with respect to $\mathcal{R}$. $\diamond$

The main result underlying the dependency pair approach is the following:

Theorem 1.48 ([AG00])**.** *Let $\mathcal{R}$ be a TRS. $\mathcal{R}$ is terminating iff the DP problem $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$ is finite.* $\square$

The condition in Definition 1.47 that terms $t_1, t_2, \ldots$ are terminating with respect to $\mathcal{R}$ corresponds to the so-called *minimality* flag in the dependency pair framework [GTSK04]. Minimality will be important in Chapter 3. However when minimality plays no role the above theorem can be stated in a simpler version.

Theorem 1.49 ([AG00])**.** *Let $\mathcal{R}$ be a TRS.* $\mathrm{SN}(\mathcal{R})$ *iff* $\mathrm{SN}(\mathrm{DP}(\mathcal{R})_{\mathrm{top}}/\mathcal{R})$*.* $\square$

Now we introduce the concept of DP processors, that is functions used to prove finiteness of DP problems.

Definition 1.50. [DP processor] A *DP processor* is a function that takes a DP problem as input and returns a set of DP problems as output.

A DP processor $(\mathcal{P}, \mathcal{R}) \mapsto \{(\mathcal{P}_i, \mathcal{R}_i)\}_{i \in \mathcal{I}}$ is *sound* if finiteness of $(\mathcal{P}_i, \mathcal{R}_i)$ for all $i \in \mathcal{I}$ implies that $(\mathcal{P}, \mathcal{R})$ is finite. $\diamond$

Below we shortly introduce two key concepts of the dependency pair method that will be important in later chapters: argument filtering and usable rules [AG00, GTSKF06].

Definition 1.51. [Argument filtering] An argument filtering is a mapping π that assigns to every n-ary function symbol f an argument position $i \in \{1,\ldots,n\}$ or a (possibly empty) list $[i_1,\ldots,i_m]$ of argument positions with $1 \leqslant i_1 < \cdots < i_m \leqslant n$. Every argument filtering π induces a mapping on terms:

$$\pi(t) = \begin{cases} t & \text{if } t \text{ is a variable,} \\ \pi(t_i) & \text{if } t = f(t_1,\ldots,t_n) \text{ and } \pi(f) = i, \\ f(\pi(t_{i_1}),\ldots,\pi(t_{i_m})) & \text{if } t = f(t_1,\ldots,t_n) \text{ and } \pi(f) = [i_1,\ldots,i_m]. \end{cases}$$

Given a binary relation $\succ$ and an argument filtering π, we write $s \succ^\pi t$ iff $\pi(s) \succ \pi(t)$. $\diamond$

Next we introduce the concept of usable rules modulo argument filtering.

Definition 1.52. [Usable rules] Let $(\mathcal{P},\mathcal{R})$ be a DP problem and π an argument filtering. We define the set of usable rules for $(\mathcal{P},\mathcal{R})$ modulo π as

$$\mathcal{U}_\pi(\mathcal{P},\mathcal{R}) = \bigcup_{s\to t\in\mathcal{P}} \mathcal{U}_\pi(t,\mathcal{R})$$

with $\mathcal{U}_\pi(t,\mathcal{R}) = \varnothing$ if t is a variable and

$$\mathcal{U}_\pi(t,\mathcal{R}) = \mathcal{R}_f \;\cup \bigcup_{l\to r\in\mathcal{R}_f} \mathcal{U}_\pi(r,\mathcal{R}\backslash\mathcal{R}_f) \;\cup \bigcup_{i:\,\pi(f)=i\,\vee\, i\in\pi(f)} \mathcal{U}_\pi(t_i,\mathcal{R}\backslash\mathcal{R}_f)$$

if $t = f(t_1,\ldots,t_n)$. $\diamond$

We illustrate the concept of usable rules on an example.

Example 1.53. *Consider the TRS $\mathcal{R}$ from Example 1.42 and its dependency pairs $\mathcal{P}$, as presented in Example 1.46. Let the argument filtering be $\pi(\mathrm{ack}^\sharp) = 1$ and $\pi(f) = [1,\ldots,n]$ for the remaining symbols $f \in \mathcal{F}^\sharp$, where n is the arity of f. Applying the above definition yields $\mathcal{U}_\pi(\mathcal{P},\mathcal{R}) = \{3,4,7\}$.* $\lhd$

We now present the reduction pair processor.

Theorem 1.54 (Reduction Pair Processor, [GTSKF06])**.** *Let $\mathcal{P}$ and $\mathcal{R}$ be TRSs. Let $(\gtrsim, >)$ be a $\mathcal{C}_\mathcal{E}$-compatible reduction pair and let π be an argument filtering. If $\mathcal{P} = \mathcal{P}_{\gtrsim^\pi} \cup \mathcal{P}_{>^\pi}$, and $\mathcal{U}_\pi(\mathcal{P},\mathcal{R}) \subseteq \gtrsim^\pi$ then the DP processor $(\mathcal{P},\mathcal{R}) \mapsto \{(\mathcal{P}\backslash\mathcal{P}_{>^\pi},\mathcal{R})\}$ is sound.* $\square$

1.4 CoLoR: Certification of Termination Proofs

In the previous section we introduced the research area of termination of rewriting. We also mentioned the recent emphasis on automation and the annual termination

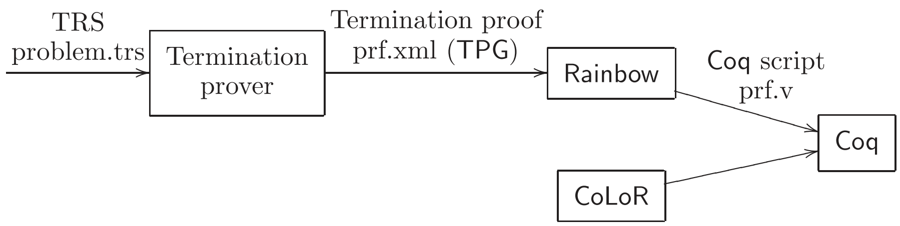


Figure 1.1 Certifying termination with CoLoR.

competition for tools proving termination automatically. An important observation from the history of this competition is that every year both the termination tools and the termination proofs produced by them are becoming more and more complex. Therefore ensuring correctness of such tools is a challenging task. Indeed every year we observe some disqualifications in the competition due to erroneous proofs produced by some of the tools.

This was one of the motivations to start the CoLoR [COL] project, initiated by Frédéric Blanqui in March 2004. The goal of the project is to use Coq [COQ, BC04], a proof assistant/checker based on the Calculus of Inductive Constructions (CIC) [PM93] (a very expressive logic supporting simple, inductive, dependant and polymorphic types), to fully automatically verify results produced by tools for proving termination. More information about the project can be found on its web-page:

`http://color.loria.fr/`

CoLoR consists of three parts:

- TPG (Termination Proofs Grammar): a formal grammar for termination proofs.
- CoLoR (Coq Library on Rewriting and Termination): a library of results on termination of rewriting, formalized in Coq.
- Rainbow: a tool for transforming termination proofs in the TPG format into Coq scripts certifying termination by employing results from CoLoR.

The general approach to certifying termination with CoLoR is presented in Figure 1.1. For a given TRS $\mathcal{R}$ some termination prover is called. If it succeeds in proving termination, it outputs a termination proof in the TPG format. Such an encoding of a proof is given to Rainbow which translates it into a Coq script containing a formal proof of the claim that $\mathcal{R}$ is terminating by using results from the CoLoR library. Then Coq is executed on such a script to verify that the termination proof found by the termination tool is indeed correct.

It is worth noting that typically Coq is used as a proof assistant, where the formalization is built by a human interacting with the system. It is not so in this application as the Coq script formalizing termination of a given system is generated fully automatically by Rainbow from a proof description produced by some termination prover; again, automatically. However the proof assistance capabilities of Coq are crucial for the development of CoLoR.

Concerning related work in certification, in the first place we should mention the Coccinelle [CCF+07] library which uses an approach similar to the one employed by CoLoR and also uses the Coq theorem prover. We will say more about it in a moment in the context of the termination competition.

The recent work at the Technical University of Munich [BKN07, Kra07] is another effort toward certified termination. It is different in several aspects. Its main aim is to automatically generate certified termination proofs for recursive functions used in the Isabelle/HOL theorem prover. However external termination provers are not involved and the only termination technique supported by this method is the size-change principle.

We already mentioned the termination competition [TC], the battlefield for termination provers. In 2007, for the first time, a new category of certified termination has been introduced, showing the recognition for the importance of certification efforts.

In this new category every claim made by a termination prover must be backed up by a full formal proof expressed and checked by some well established theorem prover (and not only by a textual informal description of such a proof, as is the case in the standard category). This makes the results reliable with the highest standards of reliability available in verification.

The combination of the CoLoR project (with Rainbow) and the termination prover TPA [Kop06c] (see Chapter 4), was the winning entry in this newly introduced category of the Termination Competition in 2007. We refer to Section 4.1.2 for more performance details.

In this competition TPA produced 354 certified proofs. For comparison we would like to mention that in the standard category, which is run on the same set of problems, the scores ranged from 330 to 723. This shows that many proofs are beyond reach of the certification at the moment, which is completely understandable. But it also shows that for a substantial part of proofs we can not only produce them with termination tools but also fully automatically ensure their correctness. We believe this is a big step forward and a very promising future for the reliability of termination results.

Chapter 2

Matrix & Arctic Interpretations

One method of proving termination is interpretation into a well-founded algebra. Polynomial interpretations (over the naturals) are a well-known example of this approach (see Section 1.3.3). Another example is the recent development of the matrix method [HW06, EWZ08] that uses linear interpretations over vectors of naturals, so functions of the shape $(\vec{x}_1, \ldots, \vec{x}_n) \mapsto M_1\vec{x}_1 + \ldots + M_n\vec{x}_n + \vec{c}$. Here, $\vec{x}_i$ are (column) vector variables, $\vec{c}$ is a vector and $M_1, \ldots, M_n$ are square matrices, where all entries are natural numbers. We will shortly introduce this method in Section 2.1.

In [Wal07b, Wal07a] the matrix interpretations method was extended (for string rewriting) to the arctic domain, *i.e.*, on the max/plus semiring on $\{-\infty\} \cup \mathbb{N}$. Its implementation in the termination prover Matchbox [Wal04] contributed to this prover winning the string rewriting division of the 2007 termination competition [TC].

In Section 2.2 we will present a *generalization of the arctic termination method to term rewriting* [KW08]. We will use interpretations of the same shape as for the matrix interpretation method, *i.e.*, $(\vec{x}_1, \ldots, \vec{x}_n) \mapsto M_1 \otimes \vec{x}_1 \oplus \ldots \oplus M_n \otimes \vec{x}_n \oplus \vec{c}$,

---

This chapter is based on: A. Koprowski, H. Zantema, Certification of Proving Termination of Term Rewriting by Matrix Interpretations, In V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat and M. Bieliková eds., *Proceedings of the 34rd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '08)*, Nový Smokovec, Slovakia, volume 4910 of Lecture Notes in Computer Science, pp. 328–339, Springer-Verlag, January 2008 and A. Koprowski, J. Waldmann, Arctic Termination ... Below Zero, In A. Voronkov ed., *Proceedings of the 19th International Conference on Rewriting Techniques and Applications, (RTA '08)*, Hagenberg, Austria, volume 5117 of Lecture Notes in Computer Science, pp. 202–216, Springer-Verlag, July 2008.

but now all the entries of vectors and matrices are arctic numbers, and operations $(\oplus, \otimes)$ are understood in the arctic semiring.

Since the max operation is not strictly monotone in single arguments, we obtain monotone interpretations only for the case when all function symbols are at most unary, *i.e.*, string rewriting. For symbols of higher arity, arctic interpretations are weakly monotone. These cannot prove termination, but only top termination, where rewriting steps are only applied at the root of terms. This is a restriction but it fits with the framework of the dependency pairs method [AG00] that transforms a termination problem to a top termination problem (see Section 1.3.7).

The second extension is a *generalization from arctic naturals to arctic integers*, *i.e.*, $\{-\infty\} \cup \mathbb{Z}$. Arctic integers allow for example to interpret function symbols by the predecessor function and this matches the "intrinsic" semantics of some termination problems. There is previous work on polynomial interpretations with negative coefficients [HM04], where the interpretation for predecessor is also expressible using ad-hoc max operations. Using arctic integers, we obtain verified termination proofs for 10 of the 24 rewrite systems Beerendonk/* from TPDB, simulating while loops. Previously, they could only be handled by the method of Bounded Increase [GTSSK07].

All those methods were formalized in the CoLoR project and we will address those formalizations in Section 2.3 [Hin04, KZ08, KW08]. We will begin by presenting the formalization of the monotone algebras framework (Section 2.3.1) and the general purpose library on matrices (Section 2.3.2) and then we will look in turn on: polynomial interpretations (Section 2.3.3), matrix interpretations (Section 2.3.4) and arctic interpretations (Section 2.3.5).

This chapter is organized as follows. We recall the theory of matrix interpretation in Section 2.1. Then we present the development of arctic interpretations in Section 2.2. Finally, in Section 2.3 we shortly introduce formalizations of those methods in CoLoR.

2.1 Matrix Interpretations

In this section we recall the theory of matrix interpretations [EWZ08]. Matrix interpretations can be seen as an instance of monotone algebras (Section 1.3.2) with $A = \mathbb{N}^d$ for some fixed dimension d. Now we define interpretations and their evaluation:

Definition 2.1. [Matrix interpretation] An *(n-ary) matrix linear function* (with linear factors $M_1, \ldots, M_n$ and an absolute part $\vec{c}$) is a function of the following shape:

$$f(\vec{x}_1, \ldots, \vec{x}_n) \ = \ M_1\vec{x}_1 + \cdots + M_n\vec{x}_n + \vec{c}$$

where $\vec{x}_1, \ldots, \vec{x}_n, \vec{c} \in \mathbb{N}^d$ and $M_1, \ldots, M_n \in \mathbb{N}^{d \times d}$.

A *matrix $\mathcal{F}$-interpretation* is an interpretation assigning an n-ary matrix linear function to every symbol $f \in \mathcal{F}$ of arity n. $\diamond$

Note that a composition of matrix linear functions yields again a matrix linear function. This is an important property as now an interpretation of an arbitrary term is a matrix linear function.

We fix a monotone algebra with $A = \mathbb{N}^d$, interpretations $[\cdot]$ defined as above and we use the orders on algebra elements defined as follows:

Definition 2.2. [Matrix orders] We define orders $\gtrsim$ and $>$ on $A = \mathbb{N}^d$ as follows:

$$\begin{aligned}(u_1, \ldots, u_d) \gtrsim (v_1, \ldots, v_d) &\iff \forall_i\ u_i \geqslant_{\mathbb{N}} v_i \\ (u_1, \ldots, u_d) > (v_1, \ldots, v_d) &\iff (u_1, \ldots, u_d) \gtrsim (v_1, \ldots, v_d) \wedge u_1 >_{\mathbb{N}} v_1\end{aligned}$$

We also make a point-wise extension of the order $\gtrsim$ to matrices. For $B, C \in \mathbb{N}^{d \times d}$ write:

$$B \gtrsim C \iff \forall_{1 \leqslant i,j \leqslant d}\ B[i,j] \gtrsim C[i,j]$$ $\diamond$

One easily checks that $(A, \{f_A\}_{f \in \mathcal{F}}, >, \gtrsim)$ is a weakly monotone $\mathcal{F}$-algebra. For an extended weakly monotone algebra, we need strict monotonicity and for that we need some restrictions on the interpretations. It is easy to see that we get strict monotonicity if for every $f \in \mathcal{F}$ with $\text{arity}(f) = n$ we require that their upper left elements $M_i[1,1]$ are positive for all matrices M_i for $i = 1, \ldots, n$.

The following lemma provides a decision procedure for comparing matrix linear interpretations, as needed for an application of Theorem 1.27:

Lemma 2.3 ([HW06])**.** *Let $[\cdot]$ be a matrix interpretation, $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $x_1, \ldots, x_n$ be the variables occurring in l, r. Then the interpretations of l and r are matrix linear functions.*

$$\begin{aligned}[l, \alpha] &= M_1 \vec{x}_1 + \ldots + M_n \vec{x}_n + \vec{c} \\ [r, \alpha] &= N_1 \vec{x}_1 + \ldots + N_n \vec{x}_n + \vec{d}\end{aligned}$$

where $\alpha(x_i) = \vec{x}_i$ for $i = 1, \ldots, n$. Then we have:

- $l \gtrsim_{\mathcal{A}} r \iff \vec{c} \gtrsim \vec{d} \wedge \forall_i\ M_i \gtrsim N_i$*, and*
- $l >_{\mathcal{A}} r \iff \vec{c} > \vec{d} \wedge \forall_i\ M_i \gtrsim N_i$. $\square$

The following two theorems are the instantiation of the monotone algebras Theorem 1.27 to matrix interpretations and provide the termination criterion for (relative) termination and (relative) top termination, respectively.

Theorem 2.4 ([HW06])**.** *Let $\mathcal{R}, \mathcal{R}', \mathcal{S}, \mathcal{S}'$ be TRSs over a signature $\mathcal{F}$ and $[\cdot]$ be matrix interpretation. If:*

- *for all* $f \in \mathcal{F}$ *with* $\text{arity}(f) = n$, $M_i[1,1] > 0$ *for* $i = 1, \ldots, n$ *and*
- $l \gtrsim_{\mathcal{A}} r$ *for all rules* $l \to r$ *in* $\mathcal{R} \cup \mathcal{S}$,
- $l >_{\mathcal{A}} r$ *for all rules* $l \to r$ *in* $\mathcal{R}' \cup \mathcal{S}'$ *and*
- $\text{SN}(\mathcal{R}/\mathcal{S})$,

then $\text{SN}(\mathcal{R} \cup \mathcal{R}'/\mathcal{S} \cup \mathcal{S}')$. □

So the approach for proving $\text{SN}(\mathcal{R}/\mathcal{S})$, using Theorem 2.4 is as follows (for proving $\text{SN}(\mathcal{R})$ this coincides with choosing $\mathcal{S} = \varnothing$):

- Fix a dimension d.
- For every symbol $f \in \mathcal{F}$ choose a vector $\vec{c} \in \mathbb{N}^d$ and matrices $M_i \in \mathbb{N}^{d \times d}$ for $i = 1, \ldots, n$ where $n = \text{arity}(f)$, such that the upper left elements of all matrices are positive.
- Check that for every rule $l \to r$ in $\mathcal{R} \cup \mathcal{S}$ we have $l \gtrsim_{\mathcal{A}} r$ using the criterion from Lemma 2.3.
- Remove all rules from $\mathcal{R}$ and $\mathcal{S}$ moreover satisfying $l >_{\mathcal{A}} r$.
- If the remaining $\mathcal{R}$ is empty we are finished since $\text{SN}(\varnothing/\mathcal{S})$ trivially holds, otherwise the process is repeated for the reduced TRSs $\mathcal{R}$, $\mathcal{S}$.

Theorem 2.5 ([HW06])**.** *Let* $\mathcal{R}, \mathcal{R}', \mathcal{S}$ *be TRSs over a signature* $\mathcal{F}$ *and* $[\cdot]$ *be matrix interpretation. If:*

- $l \gtrsim_{\mathcal{A}} r$ *for all rules* $l \to r$ *in* $\mathcal{R} \cup \mathcal{S}$,
- $l >_{\mathcal{A}} r$ *for all rules* $l \to r$ *in* $\mathcal{R}'$ *and*
- $\text{SN}(\mathcal{R}_{\text{top}}/\mathcal{S})$,

then $\text{SN}(\mathcal{R}_{\text{top}} \cup \mathcal{R}'_{\text{top}}/\mathcal{S})$. □

For proving termination of a TRS $\mathcal{R}$ by dependency pairs according to Theorem 1.48 we have to prove $\text{SN}(\text{DP}(\mathcal{R})_{\text{top}}/\mathcal{R})$. For this we apply Theorem 2.5. A similar scheme as the one for full termination, sketched above, is used, with the following difference:

- Since we only require to have a weakly monotone algebra, we may choose arbitrary matrices $M_i \in \mathbb{N}^{d \times d}$ without the restriction of positiveness of the upper left element $M_i[1,1]$.
- For proving $\text{SN}(\text{DP}(\mathcal{R})_{\text{top}}/\mathcal{R})$ only rules from $\text{DP}(\mathcal{R})$ are removed until nothing remains, and all rules from $\mathcal{R}$ are kept.

2.2 Arctic Interpretations

In this section we introduce the termination method of arctic interpretations. We begin by introducing semirings in Section 2.2.1. Then we will present the arctic method for proving full termination (restricted to string rewriting; Section 2.2.2) and top termination (Section 2.2.3) and we will present extension of the method to arctic integers (Section 2.2.3).

2.2.1 Semirings

A *commutative semiring* [Gol99] consists of a carrier D, two designated elements $d_0, d_1 \in D$ and two binary operations on D: $\oplus$ (semiring addition) and $\otimes$ (semiring multiplication), such that both $(D, d_0, \oplus)$ and $(D, d_1, \otimes)$ are commutative monoids and multiplication distributes over addition: $\forall_{x,y,z \in D}\ x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$.

One example of a semiring are the natural numbers with the standard operations, *i.e.*, natural numbers addition and multiplication. We will need the *arctic semiring* (also called the *max/plus algebra*) [GP97] with carrier $\mathcal{A}_{\mathbb{N}} \equiv \{-\infty\} \cup \mathbb{N}$, where semiring addition is the max operation with neutral element $-\infty$ and semiring multiplication is the standard plus operation with neutral element 0, so:

$$
\begin{array}{llll}
x \oplus y = y & \text{if } x = -\infty, & x \otimes y = -\infty & \text{if } x = -\infty \text{ or } y = -\infty, \\
x \oplus y = x & \text{if } y = -\infty, & x \otimes y = x + y & \text{otherwise,} \\
x \oplus y = \max(x, y) & \text{otherwise.} & &
\end{array}
$$

We also consider these operations for arctic numbers *below zero* (ie. *arctic integers*), that is, on the carrier $\mathcal{A}_{\mathbb{Z}} \equiv \{-\infty\} \cup \mathbb{Z}$.

For any semiring D, we can consider the space of linear functions (square matrices) on n-dimensional vectors over D. These functions (matrices) again form a semiring (though a non-commutative one), and indeed we write $\oplus$ and $\otimes$ for its operations as well.

A semiring is *ordered* [Fuc62] by $\geqslant$ if $\geqslant$ is a partial order compatible with the operations: $\forall_{x,y,z}\ x \geqslant y \implies x \oplus z \geqslant y \oplus z$ and $\forall_{x,y,z}\ x \geqslant y \implies x \otimes z \geqslant y \otimes z$.

The standard semiring of natural numbers is ordered by the standard $\geqslant$ relation. The semiring of arctic naturals and arctic integers is ordered by $\geqslant$, being the reflexive closure of $>$ defined as $\ldots > 1 > 0 > -1 > \ldots > -\infty$. Note that standard integers with standard operations form a semiring but it is not ordered in this sense, as we have for instance $1 \geqslant 0$ but $1 * (-1) = -1 \ngeqslant 0 = 0 * (-1)$.

We remark that $\geqslant$ is the "natural" ordering for the arctic semiring, in the following sense: $x \geqslant y \iff x = x \oplus y$. Since arctic addition is idempotent, some properties of $\geqslant$ follow easily, like the one presented below.

Lemma 2.6. *For arctic integers a_1, a_2, b_1, b_2, if $a_1 \geqslant a_2 \wedge b_1 \geqslant b_2$, then $a_1 \oplus b_1 \geqslant a_2 \oplus b_2$ and $a_1 \otimes b_1 \geqslant a_2 \otimes b_2$.* □

Arctic addition (*i.e.*, the max operation) is not strictly monotone in single arguments: we have, *e.g.*, $5 > 3$ but $5 \oplus 6 = 6 \not> 6 = 3 \oplus 6$. It is, however, "half strict" in the following sense: a strict increase in both arguments simultaneously gives a strict increase in the result, *i.e.*, $a_1 > b_1$ and $a_2 > b_2$ implies $a_1 \oplus a_2 > b_1 \oplus b_2$. There is one exception: arctic addition is obviously strict if one argument is arctic zero, *i.e.*, $-\infty$. This is the motivation for introducing the following relation:

$$a \gg b \iff (a > b) \vee (a = b = -\infty)$$

Below we present some of its properties needed later:

Lemma 2.7. *For arctic integers a, a_1, a_2, b_1, b_2,*

1. *if $a_1 \gg a_2 \wedge b_1 \gg b_2$, then $a_1 \oplus b_1 \gg a_2 \oplus b_2$.*
2. *if $a_1 \gg a_2 \wedge b_1 \geqslant b_2$, then $a_1 \otimes b_1 \gg a_2 \otimes b_2$.*
3. *if $b_1 \gg b_2$, then $a \otimes b_1 \gg a \otimes b_2$.*

Proof. By simple case analysis (whether an element is $-\infty$ or not) and some properties of addition and max operations over integers. □

Note that properties 2 and 3 in the above lemma would not hold if we were to replace $\gg$ with $>$.

An arctic natural number $a \in \mathcal{A}_{\mathbb{N}}$ is called *finite* if $a \neq -\infty$. An arctic integer $a \in \mathcal{A}_{\mathbb{Z}}$ is called *positive* if $a \geqslant 0$ (that excludes $-\infty$ and negative numbers).

Lemma 2.8. *Let $m, n \in \mathcal{A}_{\mathbb{N}}$ and $a, b \in \mathcal{A}_{\mathbb{Z}}$, then:*

1. *if m is finite and n arbitrary, then $m \oplus n$ is finite.*
2. *if a is positive and b arbitrary, then $a \oplus b$ is positive.*
3. *if m and n are finite, then $m \otimes n$ is finite.*

Proof. Direct computation. □

2.2.2 Full Arctic Termination

In this section we present a new method for proving termination of rewriting. In Section 2.1 we briefly introduced the matrix interpretations method [EWZ08]. The main difference here is that we want to use the arctic semiring for the underlying algebra operations.

We will consider vectors of arctic numbers. They form a monoid under component-wise arctic addition. Now we will construct a monotone algebra over such arctic vectors. So, for the algebra domain we choose vectors of arctic naturals, $\mathcal{A}_{\mathbb{N}}^d$, for some fixed dimension d. We define orders on arctic vectors and matrices by taking a point-wise extension of the orders $\gg$ and $\geqslant$ introduced in Section 2.2.1. We will use the same notation, *i.e.*, $\gg$ and $\geqslant$, for those lifted orders. Now we take the vector extension of $\gg$ and $\geqslant$ as, respectively, the strict and non-strict order of the algebra. Note that they are compatible, *i.e.*, $\gg \cdot \geqslant \subseteq \gg$. However with this choice we do not get well-foundedness of the strict order as $-\infty \gg -\infty$. Therefore we will restrict first components of vectors to finite elements (*i.e.*, elements different than $-\infty$, as introduced at the end of Section 2.2.1). Effectively our algebra becomes $(\mathbb{N} \times \mathcal{A}_{\mathbb{N}}^{d-1}, \{f_A\}_{f \in \mathcal{F}}, \gg, \geqslant)$.

To express functions over arctic vectors we will use arctic matrices, for which we define arctic addition and multiplication as usual. Square matrices form a non-commutative semiring with these operations. *E.g.* the 3×3 identity matrix is

$$\begin{pmatrix} 0 & -\infty & -\infty \\ -\infty & 0 & -\infty \\ -\infty & -\infty & 0 \end{pmatrix}$$

A square matrix M then maps a (column) vector $\vec{x}$ to a (column) vector $M \otimes \vec{x}$ and this mapping is linear: $M \otimes (\vec{x} \oplus \vec{y}) = M \otimes \vec{x} \oplus M \otimes \vec{y}$.

We still need to describe the class of functions used for interpretations. For that we will use linear functions such as those used for the matrix interpretation method, but now over some arctic domain, instead of natural numbers, and with the underlying operations understood in the arctic semiring; compare with Definition 2.1.

Definition 2.9. Let $\mathcal{A}$ be an arctic domain (so either arctic naturals $\mathcal{A}_{\mathbb{N}}$ or arctic integers $\mathcal{A}_{\mathbb{Z}}$). An *(n-ary) arctic linear function (over $\mathcal{A}$)* (with linear factors $M_1, \ldots, M_n$ and an absolute part $\vec{c}$) is a function of the following shape:

$$f(\vec{x}_1, \ldots, \vec{x}_n) = M_1 \otimes \vec{x}_1 \oplus \ldots \oplus M_n \otimes \vec{x}_n \oplus \vec{c}$$

So an arctic linear function over column vectors $\vec{x}_1, \ldots, \vec{x}_n \in \mathcal{A}^d$ is described by a column vector $\vec{c} \in \mathcal{A}^d$ and square matrices $M_1, \ldots, M_n \in \mathcal{A}^{d \times d}$.

We will refer to an interpretation assigning an arctic linear function for every $f \in \mathcal{F}$, as an *arctic $\mathcal{F}$-interpretation.* $\diamond$

Note that for brevity from now on we will omit the semiring multiplication sign $\otimes$ and use the following notation for arctic linear functions:

$$f(\vec{x}_1, \ldots, \vec{x}_n) = M_1\vec{x}_1 \oplus \ldots \oplus M_n\vec{x}_n \oplus \vec{c}$$

Example 2.10. *Consider a linear function:*

$$f(\vec{x}, \vec{y}) = \begin{pmatrix} 1 & -\infty \\ 0 & -\infty \end{pmatrix} \vec{x} \oplus \begin{pmatrix} -\infty & -\infty \\ 0 & 1 \end{pmatrix} \vec{y} \oplus \begin{pmatrix} -\infty \\ 0 \end{pmatrix}$$

Evaluation of this function on some exemplary arguments yields:

$$f(\begin{pmatrix} -\infty \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ -\infty \end{pmatrix}) = \begin{pmatrix} 1 & -\infty \\ 0 & -\infty \end{pmatrix} \begin{pmatrix} -\infty \\ 0 \end{pmatrix} \oplus \begin{pmatrix} -\infty & -\infty \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -\infty \end{pmatrix} \oplus \begin{pmatrix} -\infty \\ 0 \end{pmatrix} = \begin{pmatrix} -\infty \\ 1 \end{pmatrix}$$

$\lhd$

We will consider arctic linear functions over the domain of our algebra so we must make sure that evaluation of those functions stays within the domain, *i.e.*, that the first vector component is finite. The following definition and lemma address this issue.

Definition 2.11. An n-ary arctic linear function

$$f(\vec{x}_1, \ldots, \vec{x}_n) = M_1\vec{x}_1 \oplus \ldots \oplus M_n\vec{x}_n \oplus \vec{c}$$

over $\mathcal{A}_{\mathbb{N}}$ is called *somewhere finite* if:

- $\vec{c}[1]$ is finite, or
- $M_i[1, 1]$ is finite for some $1 \leqslant i \leqslant n$. $\diamond$

Lemma 2.12. *Let f be an n-ary arctic linear function over $\mathcal{A}_{\mathbb{N}}$, $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{N} \times \mathcal{A}_{\mathbb{N}}^{d-1}$ and $\vec{v} = f(\vec{x}_1, \ldots, \vec{x}_n)$. If f is somewhere finite then $\vec{v}[1]$ is finite.*

Proof.

$$f(\vec{x}_1, \ldots, \vec{x}_n)[1] = (M_1\vec{x}_1)[1] \oplus \ldots \oplus (M_n\vec{x}_n)[1]) \oplus \vec{c}[1] \tag{2.1}$$

Since f is somewhere finite we have, either:

- $\vec{c}[1]$ is finite, or
- for some $1 \leqslant i \leqslant n$, $M_i[1, 1]$ is finite but then $(M_i\vec{x}_i)[1] = M_i[1, 1]\vec{x}_i[1] \oplus \ldots \oplus M_i[1, d]\vec{x}_i[d]$, which is finite by Lemma 2.8, as $M_i[1, 1]$ is finite.

In either case one of the summands in Equation 2.1 is finite, making the whole expression finite by Lemma 2.8. $\square$

To apply the monotone algebra theorem, Theorem 1.27, we will need to compare arctic linear functions, *i.e.*, we will need some properties ensuring that, for arbitrary arguments, one arctic function always gives a vector that is greater (or greater equal) than the result of application of some other arctic functions to the same arguments.

Definition 2.13. Let f, g be arctic linear functions over $\mathcal{A}$:

$$f(\vec{x}_1, \ldots, \vec{x}_n) = M_1\vec{x}_1 \oplus \ldots \oplus M_n\vec{x}_n \oplus \vec{c}$$
$$g(\vec{x}_1, \ldots, \vec{x}_n) = N_1\vec{x}_1 \oplus \ldots \oplus N_n\vec{x}_n \oplus \vec{d}$$

We will say that f is greater (resp. greater equal) than g, notation $f \gg_\lambda g$ (resp. $f \geqslant_\lambda g$) iff:

- $c \gg d$ (resp. $c \geqslant d$) and
- $\forall_{1 \leqslant i \leqslant n}\ M_i \gg N_i$ (resp. $M_i \geqslant N_i$). ◇

We will justify the above definition in Lemma 2.15, but first we need an auxiliary result:

Lemma 2.14. *Let $M, N \in \mathcal{A}^{d \times d}$ and $\vec{x}, \vec{y} \in \mathcal{A}^d$.*

1. *If $M \gg N$ and $\vec{x} \geqslant \vec{y}$ then $M\vec{x} \gg N\vec{y}$.*
2. *If $M \geqslant N$ and $\vec{x} \geqslant \vec{y}$ then $M\vec{x} \geqslant N\vec{y}$.*

Proof. Immediate using Lemma 2.6 and the first two properties of Lemma 2.7. □

Lemma 2.15. *Let f, g be arctic linear functions over $\mathcal{A}$ and let $\vec{x}_1, \ldots, \vec{x}_n$ be arbitrary vectors.*

- *If $f \gg_\lambda g$ then $f(\vec{x}_1, \ldots, \vec{x}_n) \gg g(\vec{x}_1, \ldots, \vec{x}_n)$.*
- *If $f \geqslant_\lambda g$ then $f(\vec{x}_1, \ldots, \vec{x}_n) \geqslant g(\vec{x}_1, \ldots, \vec{x}_n)$.*

Proof. We will prove only the first case — the other one is analogous.

$$f(\vec{x}_1, \ldots, \vec{x}_n) = M_1\vec{x}_1 \oplus \ldots \oplus M_n\vec{x}_n \oplus \vec{c}$$
$$g(\vec{x}_1, \ldots, \vec{x}_n) = N_1\vec{x}_1 \oplus \ldots \oplus N_n\vec{x}_n \oplus \vec{d}$$

We have $\vec{c} \gg \vec{d}$ and $\forall_{1 \leqslant i \leqslant n}\ M_i \gg N_i$ as $f \gg_\lambda g$ and hence $M_i\vec{x}_i \gg N_i\vec{x}_i$ by Lemma 2.14. So every vector summand of the evaluation of f is related by $\gg$ with a corresponding summand of g and we conclude by Lemma 2.6. □

This finally gives us a way to compare arctic interpretations of two terms, as required in an application of Theorem 1.27. This result is given by the following lemma, which is the arctic counter-part of the absolute positiveness criterion used for polynomial interpretations [HJ98]. Also compare it with the equivalent result for matrix interpretations, Lemma 2.3.

Lemma 2.16. *Let $[\cdot]$ be an arctic $\mathcal{F}$-interpretation, $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $x_1, \dots, x_n$ be the variables occurring in l, r. The interpretations of l and r are arctic linear functions.*

$$[l, \alpha] = M_1\vec{x}_1 + \ldots + M_n\vec{x}_n + \vec{c}$$
$$[r, \alpha] = N_1\vec{x}_1 + \ldots + N_n\vec{x}_n + \vec{d}$$

where $\alpha(x_i) = \vec{x}_i$ for $i = 1, \dots, n$. Then we have:

- *If $[l, \alpha] \gg_\lambda [r, \alpha]$ then $l >_{\mathcal{A}} r$.*
- *If $[l, \alpha] \geqslant_\lambda [r, \alpha]$ then $l \gtrsim_{\mathcal{A}} r$.*

Proof. Immediate using Lemma 2.15. □

Clearly arctic linear functions are weakly monotone (because so is the max operation, *i.e.*, arctic addition) and we establish this property in the following lemma.

Lemma 2.17. *Every arctic linear function f over $\mathcal{A}$ is monotone with respect to $\geqslant$.*

Proof. Let $x_i \geqslant x_i'$. We have:

$$f(\vec{x}_1, \dots, \vec{x}_i, \dots, \vec{x}_n) = M_1\vec{x}_1 \oplus \ldots \oplus M_i\vec{x}_i \oplus \ldots \oplus M_n\vec{x}_n \oplus \vec{c}$$
$$f(\vec{x}_1, \dots, \vec{x}_i', \dots, \vec{x}_n) = M_1\vec{x}_1 \oplus \ldots \oplus M_i\vec{x}_i' \oplus \ldots \oplus M_n\vec{x}_n \oplus \vec{c}$$

All the summands are equal except for the one corresponding to the i'th argument, where we have $M_i\vec{x}_i \geqslant M_i\vec{x}_i'$ by Lemma 2.14 and we conclude

$$f(\vec{x}_1, \dots, \vec{x}_i, \dots, \vec{x}_n) \geqslant f(\vec{x}_1, \dots, \vec{x}_i', \dots, \vec{x}_n)$$

by Lemma 2.6. □

However, to obtain an extended weakly monotone algebra, and prove full termination using it, we need strict monotonicity. As remarked in Section 2.2.1, arctic addition is not strictly monotone. Hence functions introduced in Definition 2.9 are strictly monotone only if the $\oplus$ operation is essentially redundant; for instance it is immediately lost for functions of more than one argument. This essentially restricts our method to unary rewriting [TZGSK08]; a proper extension of string rewriting. As such, it was described in [Wal07a] and was applied by Matchbox in the 2007 termination competition. The following theorem provides a termination criterion for such systems. In the next section we will look at the top termination problems, which will allow us to lift this restriction and consider arbitrary TRSs.

Theorem 2.18. *Let $\mathcal{R}, \mathcal{R}', \mathcal{S}, \mathcal{S}'$ be TRSs over a signature $\mathcal{F}$ and $[\cdot]$ be an arctic $\mathcal{F}$-interpretation over $\mathcal{A}_{\mathbb{N}}$. If:*

- *every function symbol has arity at most 1,*
- *every constant $a \in \mathcal{F}$ is interpreted by $[a] = \vec{c}$ with $\vec{c}[1]$ finite,*
- *every unary symbol $s \in \mathcal{F}$ is interpreted by $[s(\vec{x})] = M \otimes \vec{x}$ with $M[1,1]$ finite,*
- $[\ell] \geqslant_\lambda [r]$ *for every rule* $\ell \to r \in \mathcal{R} \cup \mathcal{S}$,
- $[\ell] \gg_\lambda [r]$ *for every rule* $\ell \to r \in \mathcal{R}' \cup \mathcal{S}'$ *and*
- $\mathrm{SN}(\mathcal{R}/\mathcal{S})$.

Then $\mathrm{SN}(\mathcal{R} \cup \mathcal{R}'/\mathcal{S} \cup \mathcal{S}')$.

Proof. By Theorem 1.27a. Note that, by Lemma 2.15, $[\ell] \geqslant_\lambda [r]$ (resp. $[\ell] \gg_\lambda [r]$) implies $[\ell] \geqslant_\alpha [r]$ (resp. $[\ell] \gg_\alpha [r]$). So we only need to show that $(\mathbb{N} \times \mathcal{A}_{\mathbb{N}}^{d-1}, [\cdot], \gg, \geqslant)$ is an extended monotone algebra. The order $\gg$ is well-founded on this domain as with every decrease we get a decrease in the first component of the vector, which belongs to $\mathbb{N}$. Arctic functions are always weakly monotone by Lemma 2.17 and it is an easy observation that, due to the first three premises of this theorem, the interpretations that we allow here are strictly monotone. Finally we stay within the domain by Lemma 2.12 as the interpretation functions $[f]$ that we restrict to are somewhere finite (again by the first three assumptions). $\square$

We now present an example illustrating this theorem.

Example 2.19. *The relative termination problem* `SRS/Waldmann/r2` *is*

$$\{\mathrm{c\,a\,c} \to \epsilon,\ \mathrm{a\,c\,a} \to \mathrm{a}^4 \ / \ \epsilon \to \mathrm{c}^4\}$$

In the 2007 termination competition, it was solved by Jambox *[End05] via "self labeling" and by* Matchbox *via essentially the following arctic proof.*

We use the following arctic interpretation

$$[\mathrm{a}](\vec{x}) = \begin{pmatrix} 0 & 0 & -\infty \\ 0 & 0 & -\infty \\ 1 & 1 & 0 \end{pmatrix} x \qquad [\mathrm{c}](\vec{x}) = \begin{pmatrix} 0 & -\infty & -\infty \\ -\infty & -\infty & 0 \\ -\infty & 0 & -\infty \end{pmatrix} x$$

It is immediate that $[\mathrm{c}]$ *is a permutation (it swaps the second and third component of its argument vector), so* $[\mathrm{c}]^2 = [\mathrm{c}]^4$ *is the identity and we have* $[\epsilon] = [\mathrm{c}]^4$. *A short calculation shows that* $[\mathrm{a}]$ *is idempotent, so* $[\mathrm{a}] = [\mathrm{a}^4]$. *We compute*

$$[\mathrm{c\,a\,c}](\vec{x}) = \begin{pmatrix} 0 & -\infty & 0 \\ 1 & 0 & 1 \\ 0 & -\infty & 0 \end{pmatrix} \vec{x} \ [\mathrm{a\,c\,a}](\vec{x}) = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 2 & 2 & 1 \end{pmatrix} \vec{x} \ [\mathrm{a}^4](\vec{x}) = \begin{pmatrix} 0 & 0 & -\infty \\ 0 & 0 & -\infty \\ 1 & 1 & 0 \end{pmatrix} \vec{x}$$

therefore $[\mathrm{c\,a\,c}](\vec{x}) \geqslant_\lambda [\epsilon](\vec{x})$ *and* $[\mathrm{a\,c\,a}](\vec{x}) \gg_\lambda [\mathrm{a}^4](\vec{x})$*. Note also that all the top left entries of matrices are finite. This allows us to remove the strict rule* $\mathrm{a\,c\,a} \to \mathrm{a}^4$ *using Theorem 2.18. The remaining strict rule can be removed by counting letters* `a`*.* ◁

2.2.3 Arctic Top Termination

As explained earlier, there are no strictly monotone, linear arctic functions of more than one argument. Therefore in this section we change our attention from full termination to top termination problems, where only weak monotonicity is required. This is not a very severe restriction as it fits with the widely used dependency pair method that replaces a full termination problem with an equivalent top termination problem, see Section 1.3.7.

The monotone algebra that we are going to use is the same as in Section 2.2.2, *i.e.*, $(\mathbb{N} \times \mathcal{A}_{\mathbb{N}}^{d-1}, \{f_A\}_{f \in \mathcal{F}}, \gg, \geqslant)$. However now for proving top termination we will employ the second part of Theorem 1.27, so we only need a monotone algebra, instead of an extended monotone algebra. This allows us to consider arbitrary TRSs, as without the requirement of strict monotonicity we can allow arctic linear functions of more than one argument. The following theorem allows us to prove top termination in this setting:

Theorem 2.20. *Let* $\mathcal{R}, \mathcal{R}', \mathcal{S}$ *be TRSs over a signature* $\mathcal{F}$ *and* $[\cdot]$ *be an arctic* $\mathcal{F}$*-interpretation over* $\mathcal{A}_{\mathbb{N}}$*. If:*

- *for each* $f \in \mathcal{F}$*,* $[f]$ *is somewhere finite,*
- $[\ell] \geqslant_\lambda [r]$ *for every rule* $\ell \to r \in \mathcal{R} \cup \mathcal{S}$*,*
- $[\ell] \gg_\lambda [r]$ *for every rule* $\ell \to r \in \mathcal{R}'$ *and*
- $\mathrm{SN}(\mathcal{R}_{\mathrm{top}}/\mathcal{S})$*.*

Then $\mathrm{SN}(\mathcal{R}_{\mathrm{top}} \cup \mathcal{R}'_{\mathrm{top}}/\mathcal{S})$*.*

Proof. By Theorem 1.27b. By the same argument as in Theorem 2.18, $(\mathbb{N} \times \mathcal{A}_{\mathbb{N}}^{d-1}, [\cdot], \gg, \geqslant)$ is a weakly monotone algebra. So we only need to show that the evaluation stays within the algebra domain which follows from Lemma 2.12 and the first assumption. □

We will illustrate this theorem on an example now.

Example 2.21. *Consider the rewriting system* `secret05/tpa2`*:*

(1) $\mathrm{f}(\mathrm{s}(x), y) \to \mathrm{f}(\mathrm{p}(\mathrm{s}(x) - y), \mathrm{p}(y - \mathrm{s}(x)))$	(3) $\mathrm{p}(\mathrm{s}(x)) \to x$
(2) $\mathrm{f}(x, \mathrm{s}(y)) \to \mathrm{f}(\mathrm{p}(x - \mathrm{s}(y)), \mathrm{p}(\mathrm{s}(y) - x))$	(4) $x - 0 \to x$
	(5) $\mathrm{s}(x) - \mathrm{s}(y) \to x - y$

It was solved in the 2007 competition by AProVE *[GTSKF04] using narrowing followed by polynomial interpretations and by* $\mathsf{T_TT_2}$ *[KSZM] using polynomial interpretations with negative constants.*

After the DP transformation 9 dependency pairs can be removed using polynomial interpretations leaving the essential two dependency pairs:

$$\begin{aligned}&(1^\sharp)\ \mathrm{f}^\sharp(\mathrm{s}(x), y) \to \mathrm{f}^\sharp(\mathrm{p}(\mathrm{s}(x) - y), \mathrm{p}(y - \mathrm{s}(x)))\\ &(2^\sharp)\ \mathrm{f}^\sharp(x, \mathrm{s}(y)) \to \mathrm{f}^\sharp(\mathrm{p}(x - \mathrm{s}(y)), \mathrm{p}(\mathrm{s}(y) - x))\end{aligned}$$

So now, according to the dependency pair Theorem 1.49, we need to consider the relative top termination problem $\mathrm{SN}(\mathcal{R}_{\mathrm{top}}/\mathcal{S})$*, where* $\mathcal{R} = \{(1^\sharp), (2^\sharp)\}$ *and* $\mathcal{S} = \{(1), (2), (3), (4), (5)\}$*. For that consider the following arctic interpretation*

$$\begin{aligned}
[\mathrm{f}^\sharp(\vec{x}, \vec{y})] &= \begin{pmatrix} -\infty & -\infty \\ -\infty & -\infty \end{pmatrix} \vec{x} \oplus \begin{pmatrix} 0 & 0 \\ -\infty & -\infty \end{pmatrix} \vec{y} \oplus \begin{pmatrix} 0 \\ -\infty \end{pmatrix} & [0] &= \begin{pmatrix} 3 \\ 3 \end{pmatrix}\\
[\vec{x} - \vec{y}] &= \begin{pmatrix} 0 & -\infty \\ 0 & 0 \end{pmatrix} \vec{x} \oplus \begin{pmatrix} -\infty & -\infty \\ 0 & 0 \end{pmatrix} \vec{y} \oplus \begin{pmatrix} 0 \\ 0 \end{pmatrix} & [\mathrm{p}(\vec{x})] &= \begin{pmatrix} 0 & -\infty \\ 0 & -\infty \end{pmatrix} \vec{x} \oplus \begin{pmatrix} -\infty \\ -\infty \end{pmatrix}\\
[\mathrm{f}(\vec{x}, \vec{y})] &= \begin{pmatrix} 0 & 0 \\ 0 & -\infty \end{pmatrix} \vec{x} \oplus \begin{pmatrix} 2 & 0 \\ 0 & -\infty \end{pmatrix} \vec{y} \oplus \begin{pmatrix} 0 \\ -\infty \end{pmatrix} & [\mathrm{s}(\vec{x})] &= \begin{pmatrix} 0 & 0 \\ 2 & 1 \end{pmatrix} \vec{x} \oplus \begin{pmatrix} 0 \\ 2 \end{pmatrix}
\end{aligned}$$

which is somewhere finite and removes the second dependency pair:

$$\begin{aligned}
[\mathrm{f}^\sharp(\vec{x}, \mathrm{s}(\vec{y}))] &= \begin{pmatrix} -\infty & -\infty \\ -\infty & -\infty \end{pmatrix} \vec{x} \oplus \begin{pmatrix} 2 & 1 \\ -\infty & -\infty \end{pmatrix} \vec{y} \oplus \begin{pmatrix} 2 \\ -\infty \end{pmatrix}\\
[\mathrm{f}^\sharp(\mathrm{p}(\vec{x} - \mathrm{s}(\vec{y})), \mathrm{p}(\mathrm{s}(\vec{y}) - \vec{x}))] &= \begin{pmatrix} -\infty & -\infty \\ -\infty & -\infty \end{pmatrix} \vec{x} \oplus \begin{pmatrix} 0 & 0 \\ -\infty & -\infty \end{pmatrix} \vec{y} \oplus \begin{pmatrix} 0 \\ -\infty \end{pmatrix}
\end{aligned}$$

It is also weakly compatible with all the rules. The remaining dependency pair can be removed by a standard matrix interpretation of dimension two. $\triangleleft$

2.2.4 . . . Below Zero

In this section we will boldly go below zero: we extend the domain of matrix and vector coefficients from $\mathcal{A}_{\mathbb{N}}$ (arctic naturals) to $\mathcal{A}_{\mathbb{Z}}$ (arctic integers). This allows to interpret some function symbols by the "predecessor" function $x \mapsto x - 1$, and so represents their "intrinsic" semantics. This is the same motivation as the one for allowing polynomial interpretations with negative coefficients [HM04].

We need to be careful though, as the relation $\gg$ on vectors of arctic integers is not well-founded. We will solve it in a similar way as in Sections 2.2.2 and 2.2.3, that is by restricting the first component of the vectors in our domain to natural numbers, which restores well-foundedness. So we are working in the $(\mathbb{N} \times \mathcal{A}_{\mathbb{Z}}^{d-1}, \{f_A\}_{f \in \mathcal{F}}, \gg, \geqslant)$ algebra.

Again we need to make sure that we do not go outside of the domain, *i.e.*, the first vector component needs to be positive. This is ensured by the following property:

Definition 2.22. An n-ary arctic linear function

$$f(\vec{x}_1, \ldots, \vec{x}_n) = M_1 \otimes \vec{x}_1 \oplus \ldots \oplus M_n \otimes \vec{x}_n \oplus \vec{c}$$

over $\mathcal{A}_{\mathbb{Z}}$ is called *absolutely positive* if $\vec{c}\,[1]$ is positive. $\diamond$

Lemma 2.23. *Let f be an n-ary arctic linear function over $\mathcal{A}_{\mathbb{Z}}$, $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{N} \times \mathcal{A}_{\mathbb{Z}}^{d-1}$ and $\vec{v} = f(\vec{x}_1, \ldots, \vec{x}_n)$. If f is absolutely positive then $\vec{v}[1] \in \mathbb{N}$.*

Proof. Immediate, as $\vec{c}\,[1]$ positive by the definition of absolutely positive function.

$$\vec{v}[1] = f(\vec{x}_1, \ldots, \vec{x}_n)[1] = \max(\vec{c}\,[1], \ldots) \geqslant 0 \qquad \square$$

We can now present the main theorem of this section.

Theorem 2.24. *Let $\mathcal{R}, \mathcal{R}', \mathcal{S}$ be TRSs over a signature $\mathcal{F}$ and $[\cdot]$ be an arctic $\mathcal{F}$-interpretation over $\mathcal{A}_{\mathbb{Z}}$. If:*

- *for each $f \in \mathcal{F}$, $[f]$ is absolutely positive,*
- *$[\ell] \geqslant_\lambda [r]$ for every rule $\ell \to r \in \mathcal{R} \cup \mathcal{S}$,*
- *$[\ell] \gg_\lambda [r]$ for every rule $\ell \to r \in \mathcal{R}'$ and*
- *$\mathrm{SN}(\mathcal{R}_{\mathrm{top}}/\mathcal{S})$.*

Then $\mathrm{SN}(\mathcal{R}_{\mathrm{top}} \cup \mathcal{R}'_{\mathrm{top}}/\mathcal{S})$.

Proof. By Theorem 1.27b. We proved that $(\mathbb{N} \times \mathcal{A}_{\mathbb{N}}^{d-1}, \{f_A\}_{f \in \mathcal{F}}, \gg, \geqslant)$ is a weakly monotone algebra in Theorem 2.20 — now the domain is extended from arctic naturals to arctic integers but all the properties carry over easily. The fact that we respect the algebra domain is ensured by the first property and Lemma 2.23 $\square$

We now illustrate this theorem on an example.

Example 2.25. *Let us consider the* Beerendonk/2.trs *TRS from the* TPDB *[TPD], consisting of the following six rules:*

$$\begin{aligned} \mathrm{cond}(\mathrm{true}, x, y) &\to \mathrm{cond}(\mathrm{gr}(x, y), \mathrm{p}(x), \mathrm{s}(y)) & \mathrm{gr}(\mathrm{s}(x), \mathrm{s}(y)) &\to \mathrm{gr}(x, y) \\ \mathrm{gr}(0, x) &\to \mathrm{false} & \mathrm{gr}(\mathrm{s}(x), 0) &\to \mathrm{true} \\ \mathrm{p}(0) &\to 0 & \mathrm{p}(\mathrm{s}(x)) &\to x \end{aligned}$$

This is a straightforward encoding of the following imperative program

```
while x > y do (x, y) := (x-1, y+1);
```

with $x, y \in \mathbb{N}$ and the predecessor of x, i.e., $x-1$, defined on this domain, so $0-1=0$. This program is obviously terminating. However its encoding as the above TRS posed a serious challenge for the tools in the termination competition. Only AProVE *could deal with this system (as well as a number of others coming from such transformations from imperative programs) using a specialized bounded increase method [GTSSK07]. We will now show a termination proof for this system using the arctic below zero interpretation.*

We begin by applying the dependency pair method and obtaining four dependency pairs, three of which can be easily removed (for instance using standard matrix or polynomial interpretations) leaving the following single dependency pair:

$$\mathrm{cond}^\sharp(\mathrm{true}, x, y) \to \mathrm{cond}^\sharp(\mathrm{gr}(x, y), \mathrm{p}(x), \mathrm{s}(y))$$

Now, consider the following arctic matrix interpretation of dimension 1, so a degenerated case where arctic vectors and matrices simply become arctic numbers:

$$\begin{aligned}
[\mathrm{cond}^\sharp(\vec{x}, \vec{y}, \vec{z})] &= (0)\vec{x} \oplus (0)\vec{y} \oplus (-\infty)\vec{z} \oplus (0) & [0] &= (0)\\
[\mathrm{cond}(\vec{x}, \vec{y}, \vec{z})] &= (0)\vec{x} \oplus (2)\vec{y} \oplus (-\infty)\vec{z} \oplus (0) & [\mathrm{false}] &= (0)\\
[\mathrm{gr}(\vec{x}, \vec{y})] &= (-1)\vec{x} \oplus (-\infty)\vec{y} \oplus (0) & [\mathrm{true}] &= (2)\\
[\mathrm{p}(\vec{x})] &= (-1)\vec{x} \oplus (0) & [\mathrm{s}(\vec{x})] &= (2)\vec{x} \oplus (3)
\end{aligned}$$

This interpretation is absolutely positive, gives us a decrease for the dependency pair

$$\begin{aligned}
[\mathrm{cond}^\sharp(\mathrm{true}, \vec{x}, \vec{y})] &= (\ \ 0)\vec{x} \oplus (-\infty)\vec{y} \oplus (2)\\
[\mathrm{cond}^\sharp(\mathrm{gr}(\vec{x}, \vec{y}), \mathrm{p}(\vec{x}), \mathrm{s}(\vec{y}))] &= (-1)\vec{x} \oplus (-\infty)\vec{y} \oplus (0)
\end{aligned}$$

and all the original rules are oriented weakly. $\lhd$

We discuss a variant that looks more liberal, but turns out to be equivalent to the one given here. We cannot allow $\mathbb{Z} \times \mathcal{A}_{\mathbb{Z}}^{d-1}$ for the domain, because it is not well-founded for $\gg$. So we can restrict the admissible range of negative values by some bound $c > -\infty$, and use the domain $\mathcal{A}_{\mathbb{Z} \geqslant c} \times \mathcal{A}_{\mathbb{Z}}^{d-1}$ where $\mathcal{A}_{\mathbb{Z} \geqslant c} := \{b \in \mathcal{A}_{\mathbb{Z}} \mid b \geqslant c\}$. Now to ensure that we stay within this domain we would demand that the first position of the constant vector of every interpretation is greater or equal than c.

Note however that this c can be fixed to 0 without any loss of generality as every interpretation using lower values in those positions can be “shifted” upwards. For any interpretation $[\cdot]$ and arctic number d construct an interpretation $[\cdot]'$ by $[t]' := [t] \otimes d$. This is obtained by going from $[f] = M_1\vec{x}_1 \oplus \dots M_k\vec{x}_k \oplus \vec{c}$ to $[f]' = M_1\vec{x}_1 \oplus \dots M_k\vec{x}_k \oplus \vec{c} \otimes d$. (A linear function with absolute part can be scaled by scaling the absolute part.)

2.2.5 Implementation

The implementation in Matchbox follows the scheme described in [EWZ08]. The constraint problem for the arctic interpretation is translated to a constraint problem for matrices, for arctic numbers and, finally, for Boolean variables. This is then solved by Minisat [ES03].

An arctic number is represented by a pair $a = (b; v_0, v_1, \ldots, v_n)$ where b is a Boolean value and $v_0, \ldots, v_n$ is a sequence of Booleans (all numbers have fixed bit-width). If b is 1, then a represents $-\infty$, if b is 0, then a represents the binary value of $v_0, \ldots, v_n$.

To represent integers, we use two's complement representation, *i.e.*, the most significant bit is the "sign bit".

Note that implementation of max/plus operation is less expensive than standard plus/times: with a binary representation both max and plus can be computed (encoded) with a linear size formula (whereas a naive implementation of the standard multiplication requires quadratic size and asymptotically better schemes do not pay off for small bit widths).

It is useful to require the following, for each arctic number $a = (b, v)$: if the infinity bit b is set, then $v = 0$. Then $(b, v) \oplus (b', v') = (b \wedge b', \max(v, v'))$. For $(b, v) \otimes (b', v')$ we compute $c = b \vee b'$, $u = (u_0, \ldots, u_n) = v + v'$ and the result is $(c; \neg c \wedge u_0, \ldots, \neg c \wedge u_n)$.

To represent arctic integers, we use a similar convention: if the infinity flag b is set, we require that the number v represents the lowest value of its range.

The following table lists the numbers of certified proofs that we obtain with the DP transformation (without the SCC decomposition, see below) and the following matrix methods: (s)tandard, (a)rctic, below (z)ero. For comparison, we give the scores of the winners in a corresponding category of the 2007 termination competition, including, where applicable, the winner of the certified division (as all our proofs are certified). It is worth noting here that the emphasis of the arctic method is not to provide termination proofs where none were known before, but rather to provide certified (and often conceptually simpler) termination proofs where only uncertified proofs were available up to now.

Runs were executed on a single core of an Intel X5365 processor running at 3GHz. All proofs are available for inspection at the Matchbox web page [Wal04]. In all cases we used standard matrices of dimension 1 and 2 to remove rules before the DP transformation, and then matrix dimensions d from 1 up; with numbers of bit width 3, and a timeout of $1 + d^2$ seconds for each individual attempt.

It should be noted that TPA 2007 additionally used (non-linear) polynomial interpretations, and that Matchbox 2007 also used additional methods (*e.g.*, RFC match-bounds) and was running uncertified.

problem set	time	number of proofs found by the method				2007 winner	
		s	sa	sz	saz	overall	certified
975 TRS	1 min	361	376	388	389	723	354
	10 min	365	381	393	394	AProVE	TPA
517 SRS	1 min	178	312	298	320	337	N/A
	10 min	185	349	323	354	Matchbox	

Table 2.1 Performance of the matrix interpretation methods.

Here, we count only verified proofs, so we are missing about 3 to 5 proofs where Coq does not finish in reasonable time. (This happened—for exactly the same problems—also in 2007.)

For a SRS $\mathcal{R}$ we consider $\text{reverse}(\mathcal{R}) = \{\text{reverse}(l) \to \text{reverse}(r) \mid (l \to r) \in \mathcal{R}\}$. It is obvious that this transformation preserves termination both ways. Half of the allotted time is spent for each of $\mathcal{R}$ and $\text{reverse}(\mathcal{R})$. This increases the score considerably (by about one third).

The dependency pairs transformation is often combined with a decomposition of the resulting top termination problem into independent subproblems; analyzing strongly connected components of the estimated dependency graph [GAO02]. Currently, CoLoR provides only a simple graph approximation by top symbols of dependency pairs, but at the moment it is not efficient. Our current implementation therefore does not do decomposition. However, with only this simple graph approximation, this does not decrease power: note that an interpretation that removes rules from a maximal component in the DP graph (with no incoming arrows) can be extended to the complete graph by assigning constant zero to all top symbols not occurring in this component.

2.2.6 Discussion

Arctic naturals form a sub-semiring of arctic integers. So the question comes up whether Theorem 2.24 subsumes Theorem 2.20. Note that the prerequisites for both theorems are incomparable. Still there might be a method to construct from a somewhere-finite interpretation (above zero) an equivalent absolutely positive interpretation (below zero). We are not aware of any. Experience with implementation shows that it is useful to have both methods, especially for string rewriting. Naturals are easier to handle than integers because they do not require signed arithmetics. So typically we can increase the bit width or the matrix dimension for naturals. Our implementation finds several proofs according to Theorem 2.20

where it fails to find a proof according to Theorem 2.24 and vice versa.

It is interesting to ask whether the preconditions of Theorems 2.18,2.20,2.24 can be weakened. We discussed one variant at the end of Section 2.2.4. In general, a linear interpretation $[\cdot]$ with coefficients in $\mathcal{A}_{\mathbb{N}}$ ($\mathcal{A}_{\mathbb{Z}}$ respectively) is admissible for a termination proof if for each ground term t, the value $[t]$ is finite (positive, respectively). This is in fact a reachability problem for weighted (tree) automata. It is decidable for interpretations on arctic naturals, but it is undecidable for arctic integers (follows from a result of Krob [Kro92] on tropical word automata). In our setting, we do not guess an interpretation and then decide whether it is admissible. Rather, we have to formulate the decision algorithm as part of the constraint system for the interpretation. Therefore we chose sharper conditions on interpretations that imply finiteness (positiveness, respectively) and have an easy constraint encoding.

Another question is the relation of the standard matrix method to the arctic matrix method(s). Performance of our implementation suggests that neither method subsumes the other, but this may well be a problem of computing resources, as we hardly reach matrix dimension 5 and bit width 3.

As for the relation to other termination methods (*e.g.*, path orderings), the only information we have is that arctic (and other) matrix methods can do non-simple termination, while path orders and polynomial interpretations cannot; and on the other hand, the arctic matrix method implies a linear bound on derivational complexity, which is easily surpassed by path orders and other interpretations.

Max/plus polynomials have been used by Amadio [Ama05] as quasi-interpretations (*i.e.*, functions are weakly monotone), to bound the space complexity of derivations. Proving termination directly was not intended.

2.3 Certification with CoLoR

In this section we will present the formalizations of the interpretation based termination methods developed in this chapter. The formalizations are developed within the CoLoR project, see Section 1.4 for introduction to CoLoR.

The formalization of monotone algebras, the basic framework for interpretation based methods introduced in Section 1.3.2, is presented in Section 2.3.1. To deal with matrices we had to develop a Coq library of matrices; this is the subject of Section 2.3.2. Then we present instantiation of the monotone algebras framework to the methods of: polynomial interpretations (Section 2.3.3), matrix interpretations (Section 2.3.4) and two variants of arctic interpretations (Section 2.3.5).

2.3.1 Monotone Algebras

While doing the formalization of the matrix interpretations method we faced a number of design choices. The essential question was whether to simply formalize matrix interpretations as they are or to try to make the development as general as possible, such that hopefully (parts of) it could be reused for other techniques and also extensions to the technique itself would be feasible. We opted for the latter. Hence we formalized monotone algebras in their full generality and only later instantiated them to matrix interpretations; as in the theory presented in Section 2.1. This design choice paid off as this framework of monotone algebras was immediately used for polynomial interpretation and, later on, for arctic interpretations.

To achieve such a generic formalization we found the module mechanism of Coq especially useful. It allows for mass abstraction by encapsulating a number of declarations and definitions in modules. Such modules can be parameterized by means of functors, that is functions from modules to modules. For instance we formalized monotone algebras in Coq as a functor, which takes as an argument a structure describing a weakly monotone $\mathcal{F}$-algebra instance, consisting of the components listed in Definition 1.25.

For the formalization there is however one more thing that we need in order to be able to deal with concrete examples. For an application of Theorem 1.27 we need to check for arbitrary terms l and r whether $l \gtrsim_{\mathcal{A}} r$ and $l >_{\mathcal{A}} r$. Our first approach was to require the relations $>_{\mathcal{A}}$ and $\gtrsim_{\mathcal{A}}$ to be decidable, that is to require a proof that for two arbitrary elements the relation between them either holds or not. Such decidability results proven in the constructive logic of Coq provide a decision procedure. By making proofs transparent and hence allowing to reduce associated proof terms, one effectively obtains an algorithm for checking whether two given terms can be oriented with the given relation.

This approach however has one limitation: we require a decidability proof, so indeed the relations in question must be decidable. This is the case for matrix interpretations due to the characterization of Lemma 2.3 but it is not so for instance for non-linear polynomial interpretations. Therefore to make our development more general we actually require two decidable relations $\rhd$ and $\unrhd$ such that $\rhd \subseteq >_{\mathcal{A}}$ and $\unrhd \subseteq \gtrsim_{\mathcal{A}}$ and those relations are used in application of Theorem 1.27 to check whether a rule can be (weakly) oriented. The fact that they are subsets of $>_{\mathcal{A}}$ and $\gtrsim_{\mathcal{A}}$ ensures soundness of this approach. But there is no completeness requirement allowing to use some heuristics in cases where the intended relations are not decidable, such as in case of polynomial interpretations.

To give a feeling of how such theorems are stated in the theorem prover we present the Coq equivalent of Theorem 1.27a.

```
Lemma ma_relative_termination:
```

```
let S_gt := partition part_succ S in
let S_ge := partition part_succeq S in
let R_gt := partition part_succ R in
let R_ge := partition part_succeq R in
  monotone I succ ->
  snd R_ge = nil ->
  snd S_ge = nil ->
  WF (red_mod (snd S_gt) (snd R_gt)) ->
  WF (red_mod S R).
```

Let us try to explain the components of this statement. To begin with `partition P l` is a function that given a predicate `P` and a list `l`, splits this list into two parts and returns them as a pair `l1, l2`, such that `P` holds for every element of the list `l1` and does not hold for every element of `l2`.

Now `part_succ` and `part_succeq` are predicates for the `partition` function, corresponding to the relations `succ` ($>$) and `succeq` ($\gtrsim$) lifted to terms (resulting in relations $>_{\mathcal{A}}$ and $\gtrsim_{\mathcal{A}}$). We demand `succ` to be monotone, `monotone I succ`[1]. Now we require the second component of the pair `R_ge` to be empty, hence all the rules of `R` must be weakly oriented. Finally this theorem states that we can conclude `WF (red R)` if, on top of all the other requirements that we mentioned, we can prove `WF (red (snd R_gt))` so of the TRS consisting of the rules from `R` that could not be oriented strictly. Stating this problem in such "operational" style allows us to easily apply it for concrete instances of termination problems.

The monotone algebra module also contains `Coq` tactics allowing to deal with proving termination for concrete examples. This means that for using a monotone algebra approach one only needs to provide a monotone algebra instance and as a result one obtains all the results and a full machinery for proving termination. In the following sections we will sketch how we instantiated monotone algebras for the following methods:

- polynomial interpretations, Section 2.3.3,
- matrix interpretations, Section 2.3.4, and
- two variants of arctic interpretations, Section 2.3.5.

2.3.2 Matrices

To begin with, the sole fact that we had to formalize matrices may be surprising — one would expect such a general notion to be readily available in a well-established

[1]This is a requirement of extended weakly monotone algebras, however in the formalization we decided to define only weakly monotone algebras and explicitly require strict monotonicity in places where extended weakly monotone algebras are expected.

theorem prover. But it is not present in the Coq standard library. Moreover at the time of this formalization we could find only one Coq development where matrices were used: the contribution by Nicolas Magaud [Mag03], where he proves ring properties of square matrices. We decided not to use this formalization for the reasons that we discuss at the end of this section.

To implement matrices we used a generic approach by allowing entries in the matrices to be arbitrary elements from some semiring structure. For that firstly we expressed a semiring as a module type. Then we defined matrices as a functor taking as its argument such a semiring structure and as a result producing the structure of matrices of arbitrary size with entries from the semiring domain.

Internally we represent matrices as vectors of vectors. Vectors are defined in the standard library of Coq (Coq.Bool.BVector) with the type `vector A n` representing a vector of `n` elements of type `A`. Apart from this definition the Coq standard library provides only few basic properties and operations on this type. But on the other hand, building on that, the CoLoR project provides a rich set of results about vectors that were further extended in the course of this development. Here we informally define some of these functions, which we will need later on in the presentation:

$$\begin{aligned} \text{Vnth } [a_1; \ldots a_n]\ i &= a_i \\ \text{Vfold\_left } f\ [a_1; \ldots a_n]\ b &= f\, a_1\, (f \ldots (f\, a_n\, b) \ldots) \\ \text{Vmap } f\ [a_1; \ldots a_n] &= [f\, a_1; \ldots f\, a_n] \\ \text{Vmap2 } f\ [a_1; \ldots a_n]\ [b_1; \ldots b_n] &= [f\, a_1\, b_1; \ldots f\, a_n\, b_n] \end{aligned}$$

The ability to reuse those results was our main motivation to represent matrices in the following way:

```
Definition matrix (m n : nat) : matrix m n := vector (vector A n) m.
```

Then a number of operations on matrices was defined and some of its properties proven. The library is by no means complete and contains little more than the results needed for certification of matrix based methods presented in this chapter. The provided operations include: matrix creation (given matrix size and a function providing values for all matrix entries), several accessor functions to retrieve matrix elements, columns and rows, conversions from vectors to 1-row and 1-column matrices and few standard matrix operations such as transposition, addition and multiplication. To show how reusing results about vectors substantially eased our task we present below the definition of multiplication.

First we need a few auxiliary functions on matrices. We begin with three accessor functions: `get_row`, `get_col` and `get_elem` to retrieve, respectively, the i'th row, the j'th column and the element at position (i, j) of a given matrix.[2]

[2]Note that variables `m`, `n`, `i` and `j` below do not have type annotations as their types can be

```
Definition get_row m n (M : matrix m n) i (ip : i < m) := Vnth M ip.
Definition get_col m n (M : matrix m n) j (ip : j < n) :=
  Vmap (fun v => Vnth v ip) M.
Definition get_elem m n (M : matrix m n) i j (ip : i < m) (jp : j < n) :=
  Vnth (get_row M ip) jp.
```

Note that those functions are partial as indexes `i` and `j` must be within the boundaries of a matrix `M`. In `Coq` all functions are total and to deal with this we use additional arguments for those functions, the so-called domain predicates, which ensure that the arguments are within the domain of the function.

Next we introduce the `mat_build` function, which constructs a $m \times n$ matrix from two natural numbers `m` and `n`, and a function `f` which, given a matrix position, returns the value of a matrix element to be placed at that position. Again, this function `f` is partial as it is defined only for coordinates i, j such that $0 \leqslant i < m$ and $0 \leqslant j < n$.[3] Defining function `mat_build` explicitly is not an easy task due to the presence of domain predicates and dependent types. Therefore we use `Coq` proving capabilities to prove existence of such a function using its specification.[4]

```
Definition mat_build_spec m n (gen : forall i j, i < m -> j < n -> A),
  { M : matrix m n | forall i j (ip : i < m) (jp : j < n),
    get_elem M ip jp = gen i j ip jp }.
Proof. [...] Defined.
```

and we extract the computational content from the above constructive proof to obtain the required `mat_build` function.

Having all those auxiliary, general purpose functions on vectors and matrices defining matrix multiplication is fairly straightforward. First we introduce a dot product of two vectors as:

```
Definition dot_product (n : nat) (l r : vector A n) : vector A n :=
  Vfold_left Aplus A0 (Vmap2 Amult l r).
```

where `A0` is the zero element of the domain (the additive identity of the semiring) and `Aplus` is the addition. Then multiplication becomes:

```
Definition mat_mult m n p (L : matrix m n) (R : matrix n p) :=
  mat_build (fun i j ip jp => dot_product (get_row L ip) (get_col R jp)).
```

As can be seen from this example abstracting away natural operations on vectors and matrices and then using them for more complex constructs has big advantages.

inferred by `Coq` and hence can be omitted. In this case all those variables range over natural numbers as a careful reader can easily check.

[3] We index matrix rows and columns starting from 0.

[4] Please note that we are using the `Coq` mechanism of implicit arguments to skip arguments that can be inferred by `Coq` due to type dependencies. So for the function `get_elem M i j ip jp` arguments `i` and `j` can be inferred from the domain predicates `ip` and `jp`.

Not only the definitions became significantly simpler but also reasoning about them, as one can first prove properties about such auxiliary functions and then use them to reason about more complex constructs.

In fact this was the main reason against using the development by Nicolas Magaud, mentioned at the beginning of this section. It provides nice results by proving the ring properties for square matrices. But the fact that it is stand-alone and does not provide this kind of separation as mentioned above, made it difficult to use in our setting. For instance a function for matrix addition is realized there by a relatively complex Fixpoint construct (which is 16 lines long), whereas we can simply write

```
Definition vec_plus n (L R : vector A n) := Vmap2 Aplus L R.
Definition mat_plus m n (L R : matrix m n) := Vmap2 (@vec_plus n) L R.
```

and use all CoLoR properties of `Vmap2` to prove properties of matrix addition. Similarly other operations could be expressed easily and concisely by using operations and properties of vectors available in CoLoR.

2.3.3 Polynomial Interpretations

Polynomial interpretations were contributed to the CoLoR library by Sébastien Hinderer [Hin04]. By using his results we could easily construct a monotone algebra instance corresponding to polynomial interpretations method and use the monotone algebras machinery for proving termination with this method. This has the following advantages:

- Before, it was not possible to prove termination step-wise. So in order to prove termination one had to find a polynomial interpretation such that all the rules could be oriented strictly and then one could conclude termination of the whole system. The approach of monotone algebras on the other hand allows to orient strictly only some rules and, provided that the remaining rules can be weakly oriented, those strict rules can be removed and one may continue with proving termination for this simpler system. This brings a big improvement in the proving power of the method and corresponds to the way it is used in automatic termination provers.
- The development of polynomial interpretations supported only termination, $\mathrm{SN}(\mathcal{R})$. The setting of monotone algebras supports also relative termination, $\mathrm{SN}(\mathcal{R}/\mathcal{S})$, and relative-top termination, $\mathrm{SN}(\mathcal{R}_{\mathrm{top}}/\mathcal{S})$. So by expressing polynomial interpretations as an instance of the monotone algebra approach we obtained the support for treating those more general problems for free.

Instantiating the monotone algebra results for the method of polynomial interpretations was straightforward. We achieved it in mere 117 sparse lines of code and with very minor modifications to the development of Sébastien Hinderer (essentially to get the relation for orienting rules weakly).

2.3.4 Matrix Interpretations

Now we will explain how monotone algebras are instantiated for the matrix interpretation method, so we will develop the `Coq` counter-part of the theory described in Section 2.1. First we introduce a data type representing a matrix interpretation of a function symbol:

```
Variables (A : Set) (matrix : nat -> nat -> Set)
          (Sig : Signature) (f : symbol Sig) (dim : nat).

Record matrixInt (argCnt : nat) : Type := mkMatrixInt
{
  const : vector A dim;
  args : vector (matrix dim dim) argCnt
}.
```

So `matrixInt n` is a type of matrix interpretation for a function symbol of arity `n`, defined as a record with two fields: `const` being a constant vector of the interpretation of size `dim` and `args` representing coefficients for the arguments with a `dim`×`dim` matrix per argument. Comparing with Definition 2.1, `const` represents the $\vec{f}$ vector and `args` the list of matrices $F_1, \cdots, F_n$.

Now we enclose all the parameters required for the application of Theorem 2.4, in a module type:

```
Module Type TMatrixInt.
  Parameter sig : Signature.
  Parameter dim : nat.
  Parameter dim_pos : dim > 0.
  Parameter trsInt : forall f: sig, matrixInt nat nat_matrix dim (arity f).
End TMatrixInt.
```

So we take a signature `sig`, dimension for matrices (`dim`; d in Section 2.1), a proof that dimension is positive (`dim_pos`) and interpretations for all function symbols of the signature, with respective arities (`trsInt`).

Given those parameters we construct the respective monotone algebra. The most difficult property was actually decidability of algebra relations $>$ and $\gtrsim$ lifted to terms. This corresponds to proving the '$\Leftarrow$' parts of Lemma 2.3. Note that we did not prove the '$\Rightarrow$' parts of that theorem, which state completeness of this characterization and which are not needed for the correctness of the approach. Proving the '$\Leftarrow$' part required performing linearization of the computation of a matrix interpretation, such as in Lemma 2.3. Then we proved that evaluating this

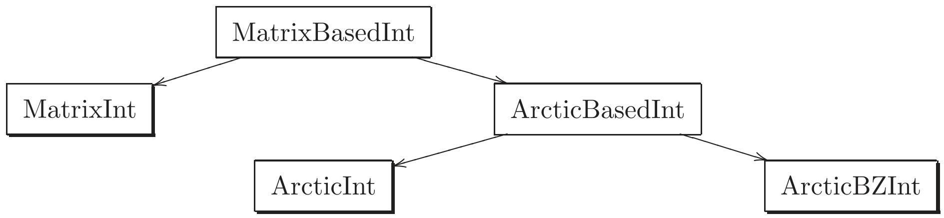


Figure 2.1 Hierarchy of different matrix-based methods in CoLoR.

linearized expression leads to the same result as simply evaluating this expression without any simplifications beforehand. Performing those two steps in Coq requires a substantial effort. For more details we refer to [KZ08].

2.3.5 Arctic Interpretations

The formalization of arctic interpretations was developed one year after the matrix interpretations. It turned out that a big part of the latter formalization, described in the previous sections, could be reused for arctic interpretations.

The framework of monotone algebras was used without any changes at all. Vectors and matrices were already formalized for arbitrary semirings, however all the results involving orders were developed for the fixed usual orders on natural numbers, as used in the matrix interpretations method. So the first step in the certification of arctic interpretations was to generalize the semiring structure to a semiring equipped with two orders $(>, \geqslant)$ and to adequately generalize results on vectors and matrices. Then the arctic semiring was developed in this setting.

As for the technique itself it has a lot in common with the technique of matrix interpretations. Therefore the previous formalization of matrix interpretations was re-factored and the common parts were extracted to a module `MatrixBasedInt`, which was then specialized to the matrix interpretation method (`MatrixInt`) and to a basis for arctic based methods (`ArcticBasedInt`), which was narrowed down to the methods of arctic interpretations (`ArcticInt`) and arctic below-zero interpretations (`ArcticBZInt`). This hierarchy is depicted in Figure 2.1.

The parameters that must be provided to prove termination with arctic interpretations are enclosed in the following module:

```
Module Type TArcticInt.
  Parameter sig : Signature.
  Parameter dim : nat.
  Parameter dim_pos : dim > 0.
  Parameter trsInt : forall f: sig, matrixInt AN matrix_AN dim (arity f).
  Parameter trsIntOk : forall f: sig, somewhere_finite (trsInt f).
End TArcticInt.
```

This module is the same as for matrix interpretations except that the domain of the interpretations (`trsInt`) is different and there is an additional component ensuring that the interpretations are somewhere finite. The module for the arctic below-zero method is analogous; again the domain is changed appropriately and the interpretations are required to be absolute positive.

Considering the extension of the proof format in Rainbow it was minimal. The format for arctic interpretations is the same as that for matrix interpretations, except that:

- it indicates which matrix-based method is to be used, denoted by different XML tags,
- the entries of vectors and matrices are from a different domain.

2.4 Conclusions

In this chapter we presented the arctic interpretation method for proving termination of term rewriting. It is based on the matrix interpretation method [EWZ08] where the usual plus/times operations on $\mathbb{N}$ are generalized to an arbitrary semiring, in this case instantiated by the arctic semiring (max/plus algebra) on $\{-\infty\} \cup \mathbb{N}$.

We also generalized this to arctic integers. This generalization allowed us to solve 10 of Beerendonk/* examples that are difficult to prove terminating and thus far could only be solved by AProVE with the Bounded Increase technique [GTSSK07], dedicated to such class of problems coming from transformations from while loops.

Our presentation of the theory is accompanied by a formalization in the Coq proof assistant. Both matrix and arctic interpretations have been formalized and by becoming part of the CoLoR project allow us to formally verify termination proofs involving those methods. With this contribution CoLoR can now certify more than half of the systems that could be proven terminating in the 2007 competition in term rewriting and essentially all (and some more) systems in the string rewriting category.

This allows us to fully automatically certify termination of non-trivial rewrite systems, such as the Zantema/z086 SRS from the TPDB [TPD]:

$$\mathrm{a\,a} \to \mathrm{c\,b} \qquad \mathrm{b\,b} \to \mathrm{c\,a} \qquad \mathrm{c\,c} \to \mathrm{b\,a}$$

Until recently termination of this innocent looking system was an open problem [RTA, Problem 104] and now not only it can be automatically proven terminating by termination tools but also that result can be warranted by Coq.

The natural way of continuing work on certification of termination is to formalize further termination techniques. Although matrix interpretations provide a very powerful base ordering, they do not subsume other orders. Consider for instance the following system:

$$\mathrm{f}(\mathrm{s}(x), \mathrm{a}) \to \mathrm{f}(x, \mathrm{f}(\mathrm{s}(x), \mathrm{b}))$$

which can be easily proven terminating with the lexicographic path order (LPO). By a simple argument one can show that matrix interpretations are not applicable here. Even more advantageous would be a formalization of the dependency pair framework [GTSK04], a modular, powerful approach to proving termination, employed by most, if not all, successful modern termination provers. This is on-going work.

Chapter 3

Semantic Labeling over Infinite Models

Semantic labeling is a transformational technique for proving termination of TRSs [Zan95, Zan03]. We introduced it in Section 1.3.5. In this chapter we will look in more detail into automation of this method for infinite models.

Automation of semantic labeling can be done straightforwardly if the models contain only finitely many elements, typically 2 or 3. For instance, this technique is used by the tools AProVE [GTSKF04], Jambox[End05], TORPA [Zan04] and TeParLa. However, if we consider labeling with infinite sets of labels, like natural numbers, some complications show up. The main difficulty is the fact that then the labeled system typically has an infinite signature and contains infinitely many rules. For that reason such variants of semantic labeling were regarded as not feasible for automation and were not used in termination tools before 2005.

A first approach to proving termination with semantic labeling over an infinite model of natural numbers was proposed in [KZ06]. A method for automation of the search for labelings and for RPO proofs for labeled, infinite systems was presented there.

This approach has later been extended and generalized in [KM07], practically subsuming the approach of [KZ06] but we present it here for historical reasons as

---

This chapter is based on: A. Koprowski, H. Zantema, Automation of Recursive Path Ordering for Infinite Labelled Rewrite Systems, In U. Furbach and N. Shankar eds., *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR '06)*, Seattle, WA, USA, volume 4130 of Lecture Notes in Computer Science, pp. 332–346, Springer-Verlag, August 2006 and A. Koprowski, A. Middeldorp, Predictive Labeling with Dependency Pairs using SAT, In F. Pfenning ed., *Proceedings of the 21st Conference on Automated Deduction (CADE '07)*, Bremen, Germany, volume 4603 of Lecture Notes in Computer Science, pp. 410–425, Springer-Verlag, July 2007.

it was the first technique to yield an automatic termination proof for the following system.

Example 3.1. *Consider the following TRS:*

$$
\begin{array}{lrcl}
(1) & \lambda(x) \circ y & \to & \lambda(x \circ (1 \star (y \circ \uparrow))) \\
(2) & (x \star y) \circ z & \to & (x \circ z) \star (y \circ z) \\
(3) & (x \circ y) \circ z & \to & x \circ (y \circ z) \\
(4) & \mathrm{id} \circ x & \to & x \\
(5) & 1 \circ \mathrm{id} & \to & 1 \\
(6) & \uparrow \circ\ \mathrm{id} & \to & \uparrow \\
(7) & 1 \circ (x \star y) & \to & x \\
(8) & \uparrow \circ\ (x \star y) & \to & y
\end{array}
$$

This system, named δ_0 in [CHR92] and essentially equivalent to system SUBST *in [HL86] describes the process of substitution in the combinatory categorical logic with 'λ' corresponding to currying, '$\circ$' to composition, '*id*' to identity, '$\star$' to pairing and '1' and '$\uparrow$' to projections.*

Termination of this system (implying termination of the process of explicit substitution in un-typed λ-calculus) is non-trivial and was the main result of [CHR92] and [HL86]. However in [Zan95, Zan03] a very simple proof was given using only semantic labeling with natural numbers followed by an application of RPO *on the transformed system.* $\lhd$

Ability to reproduce this proof completely automatically (and making this approach fruitful in general) was the main motivation of the work in [KZ06]. The method was implemented in TPA. We will present this approach in Section 3.1.

The work of [KZ06] was followed by the development of *predictive labeling* [HM06], introduced in Section 1.3.6, which aims at improving semantic labeling by weakening the quasi-model constraints — it allows to consider only *usable rules* instead of all rules of the rewrite system under consideration when checking the quasi-model condition and it requires semantics only for the relevant part of the signature.

This was the starting point for [KM07] which extends the approach of [KZ06] in a number of ways. Firstly the power of predictive labeling is enhanced by incorporation into the framework of *dependency pairs* [AG00, GTSK04]; see Section 1.3.7 for introduction to dependency pairs. This requires an extension of the theory of predictive labeling which in [HM06] was presented for ordinary termination only. Furthermore, since labeling with natural numbers produces infinite systems over infinite signatures, powerful ingredients of the dependency pair method like usable rules with argument filterings, Theorem 1.54, are not directly applicable.

The second extension is a replacement of semantic labeling with predictive labeling. This is not completely straightforward due to the fact that apart from choosing semantics for function symbols one now also needs to decide which symbols to label which in turn influences the set of usable rules. This greatly enlarges the search space.

Finally the implementation has been carried out in a different way. The corresponding search space is much larger as, instead of a small pool of predefined functions, arbitrary polynomials over natural numbers and linear functions over vectors of natural numbers are used for the semantics. Therefore, instead of a direct approach, the TPA implementation of this technique uses encoding into SAT. This means that it transforms the corresponding search problem to propositional formulas and hands them over to a SAT solver. This allows to efficiently explore much greater search space.

The approach of [KZ06] will be presented in Section 3.1. All the aforementioned extensions and the resulting termination method [KM07] will be addressed in Section 3.2. We will conclude in Section 3.3.

3.1 Automation of Semantic Labeling with Natural Numbers

In Section 1.3.4 we introduced RPO and explained that its implementation essentially consists of collecting requirements on the RPO precedence needed to orient all rules and then checking whether such precedence is well-founded – this corresponds to checking acyclicity in the corresponding graph.

For the infinite signatures, as we consider in this chapter, it is more involved, but the basic frame of the algorithm remains the same. We will concentrate on the question of how to construct a well-founded precedence satisfying a given collection of constraints.

We introduce semantic labeling with natural numbers in Section 3.1.1 and present a way of automating RPO for infinite labeled system in Section 3.1.2. In Section 3.1.3 we present practical evaluation of this techniques along with two examples.

3.1.1 Semantic Labeling with Natural Numbers

Semantic labeling was introduced in Section 1.3.5. In this section we focus on the case where $A = \mathbb{N}$, the natural numbers. We also fix the identity labeling, *i.e.*, $L_f = A^n$ and $\ell_f(x_1, \ldots, x_n) = (x_1, \ldots, x_n)$ for every $f \in \mathcal{F}$ with arity$(f) = n$.

The approach is as follows: for a TRS $\mathcal{R}$ for which we want to prove termination, search for interpretations in A such that $(A, \{f_A\}_{f \in \mathcal{F}}, >, \gtrsim)$ is a quasi-model, and next try to prove termination of the infinite TRS $\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}_{\mathcal{R}}$ by means of RPO. If this succeeds, then according to the main property of semantic labeling we have proved termination of $\mathcal{R}$. In case $(A, \{f_A\}_{f \in \mathcal{F}}, >, \gtrsim)$ happens to be a model, *i.e.*, for all rules we have equality, then we choose $\gtrsim$ on $A = \mathbb{N}$ to be equality, by which $\mathcal{D}\mathrm{ec}_{\mathcal{R}}$ is empty. In the other case we choose $>$ and $\gtrsim$ to be the usual orders on $\mathbb{N}$. In this case $\mathcal{D}\mathrm{ec}_{\mathcal{R}}$ is not empty, but we may and shall restrict $\mathcal{D}\mathrm{ec}$ to all rules of the shape

$$f_{a_1,\ldots,a_n}(x_1,\ldots,x_n) \to f_{b_1,\ldots,b_n}(x_1,\ldots,x_n)$$

for $f \in \mathcal{F}$ and $a_1,\ldots,a_n,b_1,\ldots,b_n$ satisfying $a_i = b_i + 1$ for some i and $a_j = b_j$ for all $j \neq i$. This is valid since if $a_i > b_i$ then a_i can be obtained from b_i by taking the successor a number of times, so the rewrite relation $\to^{+}_{\mathcal{D}\mathrm{ec}}$ is not changed by this modification of $\mathcal{D}\mathrm{ec}$.

Example 3.2. *As an example, we apply this approach to the TRS $\mathcal{R}$ from Example 3.1 and the following interpretation in $\mathbb{N}$:*

$$\begin{aligned} [\lambda](x) &= x+1 & \qquad [1] &= 0 \\ [\star](x,y) &= \max(x,y) & [\uparrow] &= 0 \\ [\circ](x,y) &= x+y & [\mathrm{id}] &= 0 \end{aligned}$$

This interpretation is a quasi-model and after application of semantic labeling we obtain $\mathcal{R}_{\mathsf{lab}}$ consisting of the rules

$$\begin{array}{rll}
(1) & \lambda_i(x) \circ_{i+1,j} y \to \lambda_{i+j}(x \circ_{i,j} (1 \star_{0,j} (y \circ_{j,0} \uparrow))) & \\
(2a) & (x \star_{i,j} y) \circ_{i,k} z \to (x \circ_{i,k} z) \star_{i+k,j+k} (y \circ_{j,k} z) & \textit{for } i \geqslant j \\
(2b) & (x \star_{i,j} y) \circ_{j,k} z \to (x \circ_{i,k} z) \star_{i+k,j+k} (y \circ_{j,k} z) & \textit{for } i < j \\
(3) & (x \circ_{i,j} y) \circ_{i+j,k} z \to x \circ_{i,j+k} (y \circ_{j,k} z) & \\
(4) & \mathrm{id} \circ_{0,i} x \to x & \\
(5) & 1 \circ_{0,0} \mathrm{id} \to 1 & \\
(6) & \uparrow \circ_{0,0} \mathrm{id} \to \uparrow & \\
(7a) & 1 \circ_{0,i} (x \star_{i,j} y) \to x & \textit{for } i \geqslant j \\
(7b) & 1 \circ_{0,j} (x \star_{i,j} y) \to x & \textit{for } i < j \\
(8a) & \uparrow \circ_{0,i} (x \star_{i,j} y) \to y & \textit{for } i \geqslant j \\
(8b) & \uparrow \circ_{0,j} (x \star_{i,j} y) \to y & \textit{for } i < j
\end{array}$$

and $\mathcal{D}\mathrm{ec}_{\mathcal{R}}$ consisting of the rules

$$\begin{array}{rlrl}
(D_1) & \lambda_{i+1}(x) \to \lambda_i(x) & & \\
(D_{2a}) & x \circ_{i+1,j} y \to x \circ_{i,j} y & (D_{3a}) & x \star_{i+1,j} y \to x \star_{i,j} y \\
(D_{2b}) & x \circ_{i,j+1} y \to x \circ_{i,j} y & (D_{3b}) & x \star_{i,j+1} y \to x \star_{i,j} y.
\end{array}$$

where variables i, j, k run over $\mathbb{N}$. ◁

The goal now is to represent such an infinite labeled system $\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}_{\mathcal{R}}$ in such a way that we can search systematically for a suitable RPO proving its termination. Before doing so first we say something about the search for (quasi-)models.

As long as all basic interpretations are polynomials, checking whether this interpretation is a model coincides with checking whether $[l]_\alpha = [r]_\alpha$ for all rules $l \to r$ and all α. This is simply checking whether polynomials are equal. Checking whether an interpretation is a quasi-model coincides with checking whether $[l]_\alpha \gtrsim [r]_\alpha$; this can be done along the lines of the standard way of checking for polynomial interpretations as described in [CL87, HJ98].

In early versions of TPA as a initial step symbols of arity > 2 were transformed to a number of binary symbols, so no symbols of arity > 2 would occur anymore. Then in the basic setting the functions used as interpretations for constant, unary and binary symbols, respectively, were as follows[1]:

$$\{0,\ 1\}$$
$$\{\lambda x.0,\ \lambda x.1,\ \lambda x.x,\ \lambda x.x+1,\ \lambda x.\max(0, x-1),\ \lambda x.2x,\ \lambda x.7x\}$$
$$\{\lambda x\,y.0,\ \lambda x\,y.1,\ \lambda x\,y.x+y,\ \lambda x\,y.x+y+3,\ \lambda x\,y.xy,$$
$$\lambda x\,y.x,\ \lambda x\,y.y,\ \lambda x\,y.\max(0, x-y),\ \lambda x\,y.\max(x,y),\ \lambda x\,y.\min(x,y)\}$$

So we may also want to use non-polynomial functions like min or max. Checking whether the required (in-)equalities hold is accomplished by first removing min and max functions by simple case analysis and then using the standard approach for polynomials.

Note however that while doing this case analysis we introduce side conditions, just like in Example 3.2. Let $\vec{x} = (x_1, \ldots, x_n)$. So now the problem of comparing polynomials from the standard one $\forall_{\vec{x} \in \mathbb{N}^n}\ \mathcal{P}(\vec{x}) \geqslant 0$ changes to $\forall_{\vec{x} \in \mathbb{N}^n}\ \{\mathcal{Q}_i(\vec{x}) \geqslant 0\}_{i \in \mathcal{I}} \implies \mathcal{P}(\vec{x}) \geqslant 0$, where the premise is a set of side conditions introduced by case analysis. This problem is undecidable as it is a generalization of polynomial comparison which is already undecidable. TPA uses a very simple and naive approximation of this problem and concludes the above iff $\forall_{\vec{x}}\ \mathcal{P}(\vec{x}) \geqslant 0$ or $\exists_{i \in \mathcal{I}}\ \forall_{\vec{x}}\ \mathcal{P}(\vec{x}) - \mathcal{Q}_i(\vec{x}) \geqslant 0$.Recently a more systematic and general way of dealing with polynomials with min and max was developed in [FGM$^+$08]. We illustrate our approach on an example.

Example 3.3. *Consider comparison of function symbols $\circ_{i,k}$ and $\circ_{j,k}$ in rule (2a). It may require comparing polynomials $i + k$ and $j + k$. For that we may use the side condition of this rule, $i \geqslant j$. We cannot conclude $i + k \geqslant j + k$ in general but by using side condition and subtracting i from the left hand side of this inequality and j from the right hand side we get $i + k - i \geqslant j + k - j$ which is trivially satisfied.* $\triangleleft$

[1] Note that, for technical reasons, TPA actually used $A = \mathbb{N}\backslash\{0, 1\}$ not $A = \mathbb{N}$ hence the actual functions being used are slight variants of those presented here.

3.1.2 RPO for Infinite Labeled Systems

In this section we describe how RPO can be adapted to deal with labeled systems, more precisely, for infinite systems over infinite signatures obtained by labeling with natural numbers. In fact we do not change the definition of RPO, but we restrict the search space for possible precedences on the labeled symbols in such a way that this search can be automated and we have algorithms checking whether constraints on the precedence give rise to a well-founded precedence or not.

We begin by presenting the theoretical foundations of those results and then continue with discussion of the algorithmic approach for searching for a precedence satisfying a given set of constraints.

Well-foundedness of a Precedence

The final precedence $\succ$ we search for will be of the following shape:

Definition 3.4. [Precedence description] A *precedence description* consists of:

- functions $\phi_f : \mathbb{N}^n \to \mathbb{N}$ for every $f \in \mathcal{F}$ with $\text{arity}(f) = n$ (we will call those functions *label synthesis functions*) and
- function $\mathsf{pd} : \mathcal{F} \times \mathcal{F} \to \{\bot, >, \geqslant, \top\}$.

These ingredients give rise to the following relation $\succ$:

$$\begin{aligned} f_{k_1,\dots,k_n} \succ g_{l_1,\dots,l_m} \iff & \mathsf{pd}(f,g) = \top \ \vee \\ & \big(\mathsf{pd}(f,g) = \geqslant \ \wedge \ \phi_f(k_1,\dots,k_n) \geqslant \phi_g(l_1,\dots,l_m)\big) \ \vee \\ & \big(\mathsf{pd}(f,g) = > \ \wedge \ \phi_f(k_1,\dots,k_n) > \phi_g(l_1,\dots,l_m)\big) \qquad \diamond \end{aligned}$$

So $\mathsf{pd}(f,g)$ indicates when we can conclude $f_{k_1,\dots,k_n} \succ g_{l_1,\dots,l_m}$ with: $\bot$ indicating that this can never be the case; $\top$ that it is always the case regardless of the labels of f and g; and $\geqslant$ (resp. $>$) allows us to conclude $f_{k_1,\dots,k_n} \succ g_{l_1,\dots,l_m}$ if $\phi_f(k_1,\dots,k_n)$ is greater equal (resp. strictly greater) than $\phi_g(l_1,\dots,l_m)$.

Typically this relation $\succ$ will not be an ordering as it may not be transitive but then it may be replaced by its transitive closure so by abuse of terminology we will call it a precedence.

We need some criteria under which, in the above setting, we can conclude well-foundedness of $\succ$. If these criteria hold then termination of the labeled system, and hence of the original TRS, can be concluded if $l \succ^{\mathsf{RPO}} r$ for all rules $l \to r$ in the labeled system. So our approach can be summarized as follows: collect constraints on pd and the label synthesis functions ϕ_f from the requirement that $l \succ^{\mathsf{RPO}} r$ for all rules $l \to r$, and then check whether this gives rise to a well-founded precedence $\succ$.

This well-foundedness criterion is captured by the following theorem. A function $\mathsf{pd} : \mathcal{F} \times \mathcal{F} \to \{\bot, >, \geqslant, \top\}$ gives rise to a *precedence graph* having $\mathcal{F}$ as its set of nodes, and having three kinds of directed edges:

- an *unconditional* edge from f to g if $\mathsf{pd}(f, g) = \top$, denoted by a double arrow $\Longrightarrow$;
- a *strict* edge from f to g if $\mathsf{pd}(f, g) = >$, denoted by a single arrow $\longrightarrow$;
- a *non-strict* edge from f to g if $\mathsf{pd}(f, g) = \geqslant$, denoted by a dotted arrow $\dashrightarrow$;

We can state the theorem capturing the conditions required for $\succ$ to be a well-founded precedence.

Theorem 3.5 (Well-foundedness of a precedence)**.** *In the above setting a precedence description* pd *gives rise to a well-founded precedence* $\succ$ *if every cycle in the corresponding precedence graph*

(1) contains no unconditional edges, and

(2) contains at least one strict edge.

Proof. Suppose that the precedence is not well-founded. This means that there is an infinite sequence $f_{k_1,\ldots,k_n} \succ g_{l_1,\ldots,l_m} \succ \ldots$. Every step in this reduction corresponds to an edge in the precedence graph. Since this sequence is infinite it must traverse some cycle in the precedence graph (which is finite) infinitely often. Every cycle contains only strict and non-strict edges due to (1), which gives rise to the inequalities on ϕ functions as depicted below.

$$\begin{array}{ccccccc} \cdots \longrightarrow & f & \longrightarrow & g & \longrightarrow \cdots \\ \ldots \geqslant & \phi_f(k_1, \ldots, k_n) & \geqslant & \phi_g(l_1, \ldots, l_m) & \geqslant \ldots \end{array}$$

Due to (2) at least one of those inequalities is strict which gives rise to a decreasing weight along a cycle. Hence no cycle can be traversed infinitely often. Contradiction, we conclude well-foundedness of $\succ$. $\square$

To make the approach feasible, but still applicable to interesting examples, it is natural to restrict the choice for the label synthesis functions. In TPA the choice has been made to choose ϕ to always be identity for unary symbols. For binary symbols ϕ is chosen to be one of the three functions: summation $+$, left projection π_1 and right projection π_2. This set of synthesis functions may seem quite restricted but it works reasonably well in practice whereas a bigger set would lead to a bigger search space.

Example 3.6. *One of the constraints implied by the rule*

$$\lambda_i(x) \circ_{i+1,j} y \;\rightarrow\; \lambda_{i+j}(x \circ_{i,j} (1 \star_{0,j} (y \circ_{j,0} \uparrow)))$$

where i, j *run over the naturals, will be* $\circ_{i+1,j} \succ \lambda_{i+j}$*, for all* i, j*. Since* ϕ_λ *is fixed to be the identity, according to the definition of* $\succ$ *this gives three possibilities:*

1. $\mathsf{pd}(\circ, \lambda) \;=\; \top$,
2. $\mathsf{pd}(\circ, \lambda) \;=\, \geqslant \;\wedge\; \phi_\circ(i+1, j) \geqslant i + j$ *for all* i, j,
3. $\mathsf{pd}(\circ, \lambda) \;=\, > \;\wedge\; \phi_\circ(i+1, j) > i + j$ *for all* i, j.

By re-using the algorithm for comparing polynomials it can be determined that among the three possibilities for $\phi_\circ$ *cases 2 and 3 only hold if* $\phi_\circ = +$ *and case 3 is preferred since by Theorem 3.5 strict edges are preferred over non-strict edges.* $\lhd$

Algorithm for Computing a Well-founded Precedence

In general for every pair (f, g) the RPO constraints will give rise to a list of cases. Each case consists of a choice for $\mathsf{pd}(f, g)$ being $\top$, $\geqslant$ or $>$, and in case this is not $\top$, also a choice for ϕ_f and ϕ_g. Now the question is whether for every pair (f, g) a choice can be made in such a way that conditions of Theorem 3.5 are satisfied so that this choice gives rise to a well-founded precedence. Note that for every $f \in \mathcal{F}$ in a precedence there is a single, global function ϕ_f corresponding to it. However in the search procedure we do not know what this ϕ_f should be and hence during the search we allow using different functions for comparison with different symbols and only at the end we conclude which one should be chosen for every function symbol. We will express all those choices for ϕ and pd, in, what we call, a precedence description scheme.

Definition 3.7. [Precedence description scheme]

We define a *precedence description scheme* as a function from pairs of function symbols to a set of possible implementations of ordering description on these function symbols which is either $\bot$ or $\top$ or one of $>$, $\geqslant$ accompanied with ϕ functions to be used for comparing those two symbols.

$$\mathsf{pds}(f, g) \subseteq \{\bot\} \cup (\{>, \geqslant\} \times \mathbb{N}^{\mathbb{N}^n} \times \mathbb{N}^{\mathbb{N}^m}) \cup \{\top\}$$

where $\text{arity}(f) = n$ and $\text{arity}(g) = m$.

Now we will say that a precedence description $(\{\phi_f\}_{f \in \mathcal{F}}, \mathsf{pd})$ is *compatible* with a precedence description scheme pds if:

$$\forall_{f,g \in \mathcal{F}} \begin{cases} \mathsf{pd}(f, g) = \top \implies \top \in \mathsf{pds}(f, g) \\ \mathsf{pd}(f, g) = \,\geqslant\, \implies (\geqslant, \phi_f, \phi_g) \in \mathsf{pds}(f, g) \\ \mathsf{pd}(f, g) = \,>\, \implies (>, \phi_f, \phi_g) \in \mathsf{pds}(f, g) \end{cases}$$

$\diamond$

Let us illustrate the notion of precedence description scheme on an example.

Example 3.8. *Consider the labeled system from Example 3.2. A possible branching of an application of* RPO *to that system gives the following sets of constraints for respective rules; note that for* $\circ$ *a lexicographic left-to-right status is essential.*

$$
\begin{array}{rl}
(1) & \{\circ_{i+1,j} > \lambda_{i+j},\ \circ_{i+1,j} > \circ_{i,j},\ \circ_{i+1,j} > \star_{0,j}, \\
 & \quad \circ_{i+1,j} > 1,\ \circ_{i+1,j} > \circ_{j,0},\ \circ_{i+1,j} > \uparrow\} \\
(2a) & \{\circ_{i,k} > \star_{i+k,j+k},\ \circ_{i,k} \geqslant \circ_{j,k}\} \text{ with } i \geqslant j \\
(2b) & \{\circ_{j,k} > \star_{i+k,j+k},\ \circ_{j,k} \geqslant \circ_{i,k}\} \text{ with } i < j \\
(3) & \{\circ_{i+j,k} = \circ_{i,j+k},\ \circ_{i+j,k} > \circ_{j,k}\} \\
(D_1) & \{\lambda_{i+1} > \lambda_i\} \\
(D_{2a}) & \{\circ_{i+1,j} > \circ_{i,j}\} \\
(D_{2b}) & \{\circ_{i,j+1} > \circ_{i,j}\} \\
(D_{3a}) & \{\star_{i+1,j} > \star_{i,j}\} \\
(D_{3b}) & \{\star_{i,j+1} > \star_{i,j}\}
\end{array}
$$

Now with a finite set of label synthesis functions, transformation of those constrains to a precedence description scheme can be easily accomplished. For instance for $\mathsf{pds}(\circ,\star)$ *we need to consider the following three constraints:*

$$
\circ_{i+1,j} > \star_{0,j} \qquad \circ_{i,k} > \star_{i+k,j+k} \text{ with } i \geqslant j \qquad \circ_{j,k} > \star_{i+k,j+k} \text{ with } i < j
$$

Given a finite set of label synthesis functions we can consider all possible combinations of synthesis functions for $\circ$ *and* $\star$ *and analyze the resulting polynomial constraints. In* TPA *the set of label synthesis functions for binary symbols consists of* π_1, π_2 *and* $+$. *If* $\circ$ *always bigger than* $\star$ *then we trivially have those inequalities, so* $\top \in \mathsf{pds}(\circ,\star)$. *For remaining conditional cases we easily observe that only* $+$ *is possible as a label synthesis function for* $\circ$ *whereas only projections are allowed for* $\star$. *We get* $(\geqslant,+,\pi_1) \in \mathsf{pds}(\circ,\star)$ *because* $(i+1)+j \geqslant \pi_1(0,j)$, $i+k \geqslant \pi_1(i+k,j+k)$ *and* $i<j \implies j+k \geqslant \pi_1(i+k,j+k)$. *Similarly* $(\geqslant,+,\pi_2) \in \mathsf{pds}(\circ,\star)$. *Continuing such analysis we end up with the following precedence description scheme. For all* f *and* g *for which* $\mathsf{pds}(f,g)$ *is not listed in the table below we have* $\mathsf{pds}(f,g) = \{\perp\}$.

$$
\begin{array}{ll}
\mathsf{pds}(\circ,1) = \{\top\} & \mathsf{pds}(\circ,\lambda) = \{(>,+,\mathrm{id}),\top\} \\
\mathsf{pds}(\circ,\uparrow) = \{\top\} & \mathsf{pds}(\circ,\star) = \{(\geqslant,+,\pi_1),(\geqslant,+,\pi_2),\top\} \\
\mathsf{pds}(\circ,\circ) = \{(>,+,+)\} & \mathsf{pds}(\lambda,\lambda) = \{(>,\mathrm{id},\mathrm{id})\} \\
\mathsf{pds}(\star,\star) = \{(>,+,+)\} &
\end{array}
$$

$\triangleleft$

To summarize our approach: given some TRS find an interpretation that is a (quasi-)model and label the system. Now find a RPO termination proof for that system. This proof gives rise to a number of constraints on precedence that can be transformed to a precedence description scheme pds as in Example 3.8. Now our task is to find a precedence that satisfies those constraints and is well-founded. That is we are looking for a precedence description pd which is: (a) compatible with pds and (b) satisfies conditions of Theorem 3.5.

Before presenting an appropriate algorithm we first observe that a simpler problem of finding any precedence description compatible with a given precedence description scheme is NP-complete.

Theorem 3.9. *Suppose at least three different ϕ functions are allowed. Then given a precedence description scheme* pds*, the problem of finding a precedence description* pd *compatible with* pds *is NP-complete.*

Proof. We show a reduction from the 3-coloring problem of graphs. Given undirected graph $G = (V, E)$ problem of *3-coloring of* G is to decide whether there exist:

$$f : V \to \{1, 2, 3\} \quad \text{such that} \quad \forall_{(u,v)\in E}\ f(u) \neq f(v)$$

This problem is well-known to be NP-complete.

By assumption we have (at least) three different ϕ functions, let us call them as ϕ_1, ϕ_2 and ϕ_3 and define $\Phi = \{\phi_1, \phi_2, \phi_3\}$. Let $G = (V, E)$ be an undirected graph from 3-coloring problem. We will build precedence description scheme pds in such a way that finding a precedence description compatible with pds will be equivalent with 3-coloring of G.

For every vertex $v \in V$ we introduce a function symbol f_v in $\mathcal{F}$. For every undirected edge $u - v$ we require:

$$\mathsf{pds}(f_u, f_v) = \mathsf{pds}(f_v, f_u) = \{(>, \phi, \phi') \mid \phi \neq \phi' \in \Phi\}$$

This ensures that two function symbols corresponding to connected vertices in G get different ϕ functions. If we can find a precedence compatible with this precedence description scheme then the assignment of ϕ functions to function symbols easily translates to the assignment of colors to graph vertices satisfying 3-coloring problem constraints. □

Before we present the algorithm let us put the problem in a more practical light by discussing it in the context in which it occurs in TPA. As mentioned before TPA uses precisely three different ϕ functions for binary symbols meaning that we are on the border of conditions posted in Theorem 3.9. If it had two the problem would correspond to 2-coloring which can be solved in polynomial time. But we believe that all three functions are important and we do not want to get rid of any of them. Moreover we are about to describe an algorithm that in practice performs very well and takes negligible time in the whole search procedure.

First let us observe that from the precedence description scheme pds we can already determine the structure of the precedence graph for a precedence description pd compatible with pds. Let us note that due to the construction of the precedence description scheme for any f and g we either have $\mathsf{pds}(f,g) = \{\bot\}$ or $\top \in \mathsf{pds}(f,g) \wedge \bot \notin \mathsf{pds}(f,g)$. Now if $\mathsf{pds}(f,g) = \{\bot\}$ then we can simply choose $\mathsf{pd}(f,g) = \bot$ and hence there is no edge from f to g in the precedence graph. Otherwise f and g are connected with an edge although at this point we do not know yet what is the type of this edge.

The key to get an efficient algorithm is the observation that we can detect strongly connected components (SCCs) in the precedence graph and treat them separately. Since precedence graphs are typically spare, meaning that SCCs are small, by doing so we increase the efficiency greatly. Note that all the edges between f and g belonging to different SCCs do not lie on any cycle and hence cannot violate conditions of the Theorem 3.5. Thus they can safely be changed to unconditional edges (as then $\top \in \mathsf{pds}(f,g)$). On the other hand there cannot be an unconditional edge connecting two nodes from the same SCC since all the edges within SCC belong to some cycle, so all such options can be dropped from the precedence description scheme.

So now we can localize further reasoning to a single SCC and we can limit to strict and non-strict edges only. We still need to find ϕ functions for all function symbols and the appropriate ordering of those symbols.

Firstly for all function symbols we compute their predecessors and successors in the precedence graph:

$$\begin{aligned}\mathrm{IN}_f &= \{g \mid \mathsf{pds}(g,f) \neq \{\bot\},\ g \text{ in the same SCC as } f\}\\ \mathrm{OUT}_f &= \{g \mid \mathsf{pds}(f,g) \neq \{\bot\},\ g \text{ in the same SCC as } f\}\end{aligned}$$

Now we can compute possible label synthesis functions for every function symbol:

$$\begin{aligned}\mathrm{PSF}_f = &\bigcap_{g \in \mathrm{OUT}_f} \{\phi_f \mid (\geqslant / >, \phi_f, \phi_g) \in \mathsf{pds}(f,g)\} \cap \\ &\bigcap_{g \in \mathrm{IN}_f} \{\phi_f \mid (\geqslant / >, \phi_g, \phi_f) \in \mathsf{pds}(g,f)\}\end{aligned}$$

If for any f, $\mathrm{PSF}_f = \varnothing$ then we can finish with a negative answer. Otherwise we refine ps in the following way:

$$\mathsf{pds}'(f,g) := \{(\geqslant / >, \phi_f, \phi_g) \in \mathsf{pds}(f,g) \mid \phi_f \in \mathrm{PSF}_f,\ \phi_g \in \mathrm{PSF}_g\}$$

We continue this procedure as long as there are some changes in the refinement of pds. If at any point for any f and g, $\mathsf{pds}(f,g) = \varnothing$ we finish with negative answer. Hopefully we arrive at pds with all entries being singletons in which case we have only one potential solution; otherwise we need to consider all the possible choices. At the end we check whether condition (2) of Theorem 3.5 is satisfied that is whether there are no cycles containing non-strict edges only.

The summary of the whole procedure follows.

Definition 3.10. [Algorithm for finding a precedence description compatible with given precedence schema]

(1) Compute SCCs in the precedence graph. Refine pds in the following way:

$$\mathsf{pds}'(f,g) := \begin{cases} \{\top\} & \text{if } f \text{ and } g \text{ belong to different SCC} \\ \{s \in \mathsf{pds}(f,g) \mid s \neq \top\} & \text{otherwise} \end{cases}$$

(2) If for any pair f and g, $\mathsf{pds}(f,g) = \varnothing$ answer NO and stop.

(3) For every function symbol f compute:

$$\mathrm{IN}_f = \{g \mid \mathsf{pds}(g,f) \neq \{\bot\},\ \text{g in the same SCC as f}\}$$
$$\mathrm{OUT}_f = \{g \mid \mathsf{pds}(f,g) \neq \{\bot\},\ \text{g in the same SCC as f}\}$$

(4) For every SCC:

(4a) Compute possible label synthesis functions for every function symbol:

$$\mathrm{PSF}_f = \bigcap_{g \in \mathrm{OUT}_f} \{\phi_f \mid (\geqslant / >, \phi_f, \phi_g) \in \mathsf{pds}(f,g)\} \cap \bigcap_{g \in \mathrm{IN}_f} \{\phi_f \mid (\geqslant / >, \phi_g, \phi_f) \in \mathsf{pds}(g,f)\}$$

(4b) Refine pds:

$$\mathsf{pds}'(f,g) := \{(\geqslant / >, \phi_f, \phi_g) \in \mathsf{pds}(f,g) \mid \phi_f \in \mathrm{PSF}_f,\ \phi_g \in \mathrm{PSF}_g\}$$

(4c) If for any f and g, $\mathsf{pds}'(f,g) = \varnothing$ answer NO and stop the whole procedure.

(4d) If $\mathsf{pds}' \neq \mathsf{pds}$ set $\mathsf{pds} := \mathsf{pds}'$ and go to (4a), otherwise continue with (4e).

(4e) Consider all possible precedences compatible with pds and for each one of them check whether it complies with condition (2) from Theorem 3.5. If none does, answer NO and stop. Otherwise continue with step (4) with the next SCC or with step (5) if there are no more SCCs.

(5) Combine solutions for all SCCs to get a precedence description compatible with pds giving rise to a well-founded precedence. ◇

3.1.3 Examples

The technique of semantic labeling with natural numbers has been implemented in TPA. For more information on the tool itself and the experimental results obtained with it we refer to Chapter 4. Below we present two examples and show that using our approach both those systems can be easily proved terminating. Indeed in the termination competition of 2006, TPA succeeded in producing termination proofs for them whereas, at that time, no other termination tool could deal with those systems.

But first we would like to evaluate the claim made in Section 3.1.2, namely that typically SCCs in the precedence graph are small and hence our algorithm is efficient. On a collection of 115 TRSs for which semantic labeling with natural numbers was applicable we calculated the size of the biggest SCC occurring in the analysis of a system (note that often many different labelings are tried resulting in many applications of the algorithm from Definition 3.10). The average of those values was less than 5 confirming the claim that in practice SCCs are very small. Also the time spent on the execution of the algorithm in question on average summed up to less than 1% of the total running time of TPA.

Now we will finish the analysis of the SUBST system introduced in Example 3.1.

Example 3.11. *Let us continue with Example 3.8, where we presented a precedence description scheme for the SUBST system. Below we depict the precedence graph corresponding to this scheme.*

$$\begin{tikzcd}
\star \arrow[loop left] & \circ \arrow[l] \arrow[r] \arrow[loop above] \arrow[dl, Rightarrow] \arrow[dr, Rightarrow] & \lambda \arrow[loop right] \\
1 & & \uparrow
\end{tikzcd}$$

Using algorithm from Definition 3.10 we first observe that all nodes are in separate SCCs. So first we replace all edges between different nodes by unconditional edges. Then we are left with no choice and we end up with the following precedence:

$$\begin{aligned}
\mathsf{pd}(\circ, \lambda) &= \top \\
\mathsf{pd}(\circ, \star) &= \top \\
\mathsf{pd}(\circ, 1) &= \top & \qquad \phi_\circ &= + \\
\mathsf{pd}(\circ, \uparrow) &= \top & \qquad \phi_\lambda &= \mathrm{id} \\
\mathsf{pd}(\circ, \circ) &= > & \qquad \phi_\star &= + \\
\mathsf{pd}(\lambda, \lambda) &= > \\
\mathsf{pd}(\star, \star) &= >
\end{aligned}$$

which can be written down as:

$$
\begin{array}{llll}
\circ_{i,j} > \lambda_k & \text{for all } i,j,k & \circ_{i,j} > \circ_{k,l} & \text{if } i+j > k+l \\
\circ_{i,j} > \star_{k,l} & \text{for all } i,j,k,l & \lambda_i > \lambda_k & \text{if } i > k \\
\circ_{i,j} > 1 & \text{for all } i,j & \star_{i,j} > \star_{k,l} & \text{if } i+j > k+l \\
\circ_{i,j} > \uparrow & \text{for all } i,j & &
\end{array}
$$

One can easily check that all the rules of the labeled SUBST *TRS, see Example 3.2 can be oriented using* RPO *with this precedence. This is also essentially the same solution as presented in [Zan03].*

It is worth noting that recently a new, conceptually simpler proof was presented in [FGM+08]. It does not use the technique of semantic labeling but only polynomial interpretations extended with the min *and* max *operations. It also presents a more systematical way of dealing with the constraints arising from an analysis of interpretations involving those functions.* $\lhd$

We present one more example: a very natural TRS for which our technique succeeds.

Example 3.12. *Consider the following TRS describing a GCD (Greatest common divisor) computation in a straightforward way.*

$$
\begin{array}{rcl}
\min(x,0) \to 0 & & \max(x,0) \to x \\
\min(0,y) \to 0 & & \max(0,y) \to y \\
\min(\mathrm{s}(x),\mathrm{s}(y)) \to \mathrm{s}(\min(x,y)) & & \max(\mathrm{s}(x),\mathrm{s}(y)) \to \mathrm{s}(\max(x,y)) \\
\mathrm{gcd}(\mathrm{s}(x),0) \to s(x) & & x - 0 \to x \\
\mathrm{gcd}(0,s(y)) \to s(y) & & \mathrm{s}(x) - \mathrm{s}(y) \to x - y
\end{array}
$$
$$
\mathrm{gcd}(\mathrm{s}(x),\mathrm{s}(y)) \to \mathrm{gcd}(\max(x,y) - \min(x,y), \mathrm{s}(\min(x,y)))
$$

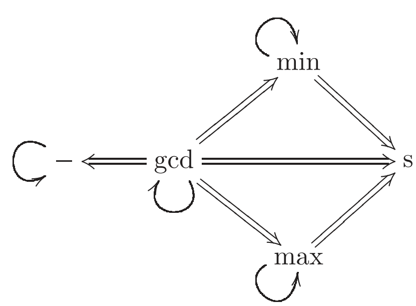
Consider the following interpretation of function symbols in $\mathbb{N}$*:*

$$
\begin{array}{rcl}
[\mathrm{s}](x) = x+1 & & [0] = 0 \\
[\min](x,y) = \min(x,y) & & [\max](x,y) = \max(x,y) \\
[-](x,y) = x & & [\mathrm{gcd}](x,y) = x+y
\end{array}
$$

This interpretation is a quasi-model and after application of semantic labeling it *gives the following TRS:*

$$
\begin{aligned}
\min_{i,0}(x,0) &\to 0 \\
\min_{0,j}(0,y) &\to 0 \\
\min_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) &\to \mathrm{s}_j(\min_{i,j}(x,y)) && \text{for } i \geqslant j \\
\min_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) &\to \mathrm{s}_i(\min_{i,j}(x,y)) && \text{for } i < j \\
\max_{i,0}(x,0) &\to x \\
\max_{0,j}(0,y) &\to y \\
\max_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) &\to \mathrm{s}_i(\max_{i,j}(x,y)) && \text{for } i \geqslant j \\
\max_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) &\to \mathrm{s}_j(\max_{i,j}(x,y)) && \text{for } i < j \\
x -_{i,0} 0 &\to x \\
\mathrm{s}_i(x) -_{i+1,j+1} \mathrm{s}_j(y) &\to x -_{i,j} y \\
\mathrm{gcd}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) &\to \mathrm{gcd}_{i-j,j}(\max_{i,j}(x,y) -_{i,j} \min_{i,j}(x,y), && \text{for } i \geqslant j \\
&\qquad \mathrm{s}_j(\min_{i,j}(x,y))) \\
\mathrm{gcd}_{i+1,j+1}(\mathrm{s}_i(x),\mathrm{s}_j(y)) &\to \mathrm{gcd}_{j-i,i}(\max_{i,j}(x,y) -_{j,i} \min_{i,j}(x,y), && \text{for } i < j \\
&\qquad \mathrm{s}_j(\min_{i,j}(x,y)))
\end{aligned}
$$

Termination of the union of the above rules and decreasing rules can be proved with RPO*. This time the description scheme leaves more choice for the label synthesis functions for different symbols but again all nodes are in separate SCCs and our algorithm produces the following precedence graph:*



and the following label synthesis functions:

$$
\begin{aligned}
\phi_{min} &= \pi_1 & \phi_s &= \mathrm{id} \\
\phi_{max} &= \pi_1 & \phi_- &= \mathrm{id} \\
\phi_{gcd} &= +
\end{aligned}
$$

Those choices correspond to the following well-founded precedence:

$$\begin{array}{rlrl}
\mathrm{min}_{i,j} > \mathrm{min}_{k,l} & \textit{if } i > k & \mathrm{gcd}_{i,j} > \mathrm{gcd}_{k,l} & \textit{if } i + j > k + l \\
\mathrm{min}_{i,j} > \mathrm{s}_k & \textit{for all } i, j, k & \mathrm{gcd}_{i,j} > \mathrm{min}_{k,l} & \textit{for all } i, j, k, l \\
\mathrm{max}_{i,j} > \mathrm{max}_{k,l} & \textit{if } i > k & \mathrm{gcd}_{i,j} > \mathrm{s}_k & \textit{for all } i, j, k \\
\mathrm{max}_{i,j} > \mathrm{s}_k & \textit{for all } i, j, k & \mathrm{gcd}_{i,j} > \mathrm{max}_{k,l} & \textit{for all } i, j, k, l \\
-_{i,j} > -_{k,l} & \textit{if } i > k & \mathrm{gcd}_{i,j} > -_{k,l} & \textit{for all } i, j, k, l
\end{array} \qquad \triangleleft$$

3.2 Automation of Predictive Labeling with Dependency Pairs using SAT

In Section 3.1 we have presented a direct approach for proving termination with semantic labeling over natural numbers, involving infinite systems. In this section we will extend this approach in several ways:

- we will use predictive labeling instead of semantic labeling (see Section 1.3.5 and Section 1.3.6 for introduction to semantic and predictive labeling) to weaken the quasi-model constraints,
- to improve the power of the method we will incorporate predictive labeling to the dependency pairs setting (see Section 1.3.7 for introduction to dependency pairs),
- instead of a preselected (small) pool of basic functions used for symbol semantics, we will use generic polynomials with arbitrary (bounded) coefficients,
- apart from polynomials over natural numbers, used for symbol semantics in labeling, we will also use linear functions over vectors of natural numbers (corresponding to the matrix interpretation method, presented in Chapter 2),
- instead of a direct approach we will use encoding to a satisfiability problem in order to boost performance and deal with much larger search space.

3.2.1 Predictive Labeling and Dependency Pairs

Predictive labeling was introduced in Section 1.3.6. In this section our goal is to extend it for the dependency pairs setting.

Before presenting this extension we state a straightforward generalization of the semantic labeling method, introduced in Section 1.3.5, to DP problems. The only observation is that interpretations of dependency pair symbols do not contribute to labels so by choosing them to be the same constant we get the quasi-model constraints for DP rules for free. We omit the easy proof.

Theorem 3.13. *Let $\mathcal{R}$ be a TRS, $(A, \{f_A\}_{f \in \mathcal{F}}, >, \gtrsim)$ a weakly monotone quasi-model for $\mathcal{R}$ and $\mathcal{L}$ a weakly monotone labeling for $\mathcal{R}$. The DP problem $(\mathrm{DP}(\mathcal{R}), \mathcal{R})$ is finite iff the DP problem $(\mathrm{DP}(\mathcal{R})_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})$ is finite.* □

The following theorem provides a DP processor for predictive labeling and constitutes the main theoretical result of this section.

Theorem 3.14. *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$ and $\mathcal{P}$ be a TRS over $\mathcal{F} \cup \mathcal{F}_{\mathrm{top}}$, such that all the non-variable terms consisting rules of $\mathcal{P}$ have a symbol from $\mathcal{F}_{\mathrm{top}}$ as a root and only symbols from $\mathcal{F}$ below the root.*

Let $(A, \{f_A\}_{f \in \mathcal{F}}, >, \gtrsim)$ be a weakly monotone $\sqcup$-algebra and $\mathcal{L}$ a weakly monotone labeling for $\mathcal{R}$ such that $\mathcal{U}(\ell) \subseteq\ \gtrsim_{\mathcal{A}}$. If $\mathcal{R}$ is finitely branching then the DP processor

$$(\mathcal{P}, \mathcal{R}) \mapsto \{(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})\}$$

is sound.

Before presenting a proof sketch, we make some clarifying remarks. The conditions on $\mathcal{P}$ ensure that the root symbols of the rules in $\mathcal{P}$ occur nowhere else. This implies that we do not have to worry about the semantics of the rules of $\mathcal{P}$ and thus $\mathcal{U}(\ell)$ will be a subset of $\mathcal{R}$. Nevertheless, the symbols from $\mathcal{F}_{\mathrm{top}}$ can be labeled and this may influence the usable rules. Note that this condition is trivially satisfied if $\mathcal{P} \subseteq \mathrm{DP}(\mathcal{R})$, which will be the case in our applications of this theorem. It follows that the definition of $\mathcal{U}(\ell)$, Definition 1.41, has to be slightly modified: $\mathcal{U}(\ell) = \{l \to r \in \mathcal{R} \mid \mathrm{root}(l) \in \mathcal{G}_\ell(\mathcal{P} \cup \mathcal{R})\}$. The $\mathcal{D}$ec rules are computed for all labeled symbols in $\mathcal{F}_{\mathrm{top}} \cup \mathcal{F}$.

Proof sketch. Suppose the DP processor $(\mathcal{P}, \mathcal{R}) \mapsto \{(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})\}$ is not sound. So the DP problem $(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})$ is finite whereas $(\mathcal{P}, \mathcal{R})$ is not. Hence there exists an infinite sequence

$$t_1 \to_{\mathcal{R}}^* u_1 \xrightarrow{\epsilon}_{\mathcal{P}} t_2 \to_{\mathcal{R}}^* u_2 \xrightarrow{\epsilon}_{\mathcal{P}} \cdots$$

such that the terms t_1, t_2, ... are terminating with respect to $\mathcal{R}$. Let α be an arbitrary assignment. We recall the following definitions from [HM06].

- Let $t \in \mathcal{SN}$. The interpretation $[\alpha]_{\mathcal{A}}^*(t)$ is inductively defined as follows:

$$[\alpha]_{\mathcal{A}}^*(t) = \begin{cases} \alpha(t) & \text{if } t \text{ is a variable,} \\ f_{\mathcal{A}}([\alpha]_{\mathcal{A}}^*(t_1), \ldots, [\alpha]_{\mathcal{A}}^*(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } f \in \mathcal{G}_\ell, \\ \bigsqcup \{[\alpha]_{\mathcal{A}}^*(u) \mid t \to_{\mathcal{R}}^+ u\} & \text{if } t = f(t_1, \ldots, t_n) \text{ and } f \notin \mathcal{G}_\ell. \end{cases}$$

- Let $t \in \mathcal{SN} \cup \mathcal{T}_\infty$. The labeled term $\mathsf{lab}^*_\alpha(t)$ is inductively defined as follows:

$$\mathsf{lab}^*_\alpha(t) = \begin{cases} t & \text{if } t \text{ is a variable,} \\ f(\mathsf{lab}^*_\alpha(t_1), \ldots, \mathsf{lab}^*_\alpha(t_n)) & \text{if } L_f = \varnothing, \\ f_a(\mathsf{lab}^*_\alpha(t_1), \ldots, \mathsf{lab}^*_\alpha(t_n)) & \text{if } L_f \neq \varnothing \end{cases}$$

  where $a = \ell_f([\alpha]^*_{\mathcal{A}}(t_1), \ldots, [\alpha]^*_{\mathcal{A}}(t_n))$.

- Given a substitution σ such that $\sigma(x) \in \mathcal{SN}$ for all variables x, the assignment α^*_σ is defined as $[\alpha]^*_{\mathcal{A}} \circ \sigma$ and the substitution $\sigma_{\mathsf{lab}^*_\alpha}$ as $\mathsf{lab}^*_\alpha \circ \sigma$.

We will apply $\mathsf{lab}^*_\alpha(\cdot)$ to the terms in the above sequence. Fix $i \geqslant 1$. Repeated application of Lemma 17 in [HM06] yields $\mathsf{lab}^*_\alpha(t_i) \to^*_{\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}} \mathsf{lab}^*_\alpha(u_i)$. We have $u_i = l\sigma$ and $t_{i+1} = r\sigma$ for some $l \to r \in \mathcal{P}$. We use Lemma 15 in [HM06] to obtain

$$\mathsf{lab}^*_\alpha(l\sigma) \to^*_{\mathcal{D}\mathrm{ec}} \mathsf{lab}_{\alpha^*_\sigma}(l)\sigma_{\mathsf{lab}^*_\alpha}.$$

Since $\mathsf{lab}_{\alpha^*_\sigma}(l) \to \mathsf{lab}_{\alpha^*_\sigma}(r) \in \mathcal{P}_{\mathsf{lab}}$, $\mathsf{lab}_{\alpha^*_\sigma}(l)\sigma_{\mathsf{lab}^*_\alpha} \xrightarrow{\epsilon}_{\mathcal{P}_{\mathsf{lab}}} \mathsf{lab}_{\alpha^*_\sigma}(r)\sigma_{\mathsf{lab}^*_\alpha}$. A variation of Lemma 16 in [HM06] gives $\mathsf{lab}_{\alpha^*_\sigma}(r)\sigma_{\mathsf{lab}^*_\alpha} = \mathsf{lab}^*_\alpha(r\sigma)$. Putting things together yields $\mathsf{lab}^*_\alpha(t_i) \to^*_{\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}} \cdot \xrightarrow{\epsilon}_{\mathcal{P}_{\mathsf{lab}}} \mathsf{lab}^*_\alpha(t_{i+1})$. Hence, the above infinite sequence is transformed into

$$\mathsf{lab}^*_\alpha(t_1) \to^*_{\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}} \cdot \xrightarrow{\epsilon}_{\mathcal{P}_{\mathsf{lab}}} \mathsf{lab}^*_\alpha(t_2) \to^*_{\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}} \cdot \xrightarrow{\epsilon}_{\mathcal{P}_{\mathsf{lab}}} \cdots$$

If we can show that the terms $\mathsf{lab}^*_\alpha(t_1)$, $\mathsf{lab}^*_\alpha(t_2)$, $\cdots$ are terminating with respect to $\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}$ then the DP problem $(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})$ is not finite, providing the desired contradiction. Suppose $\mathsf{lab}^*_\alpha(t_i)$ for some i admits an infinite reduction with respect to $\mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec}$. Because $\mathcal{D}\mathrm{ec}$ is a terminating TRS, there must be infinitely many $\mathcal{R}_{\mathsf{lab}}$-steps in this sequence. If we remove all labels, the $\mathcal{D}\mathrm{ec}$-steps disappear and the $\mathcal{R}_{\mathsf{lab}}$-steps are turned into $\mathcal{R}$-steps. It follows that t_i is not terminating with respect to $\mathcal{R}$. This completes the proof. □

After labeling we want to apply the reduction pair processor of the dependency pairs framework, see Theorem 1.54. However, originally this processor is stated only for finite TRSs and labeled systems will usually be infinite. Therefore below we present an extension to infinite, but finitely branching systems.

Theorem 3.15 (Reduction Pair Processor)**.** *Let $\mathcal{P}$ and $\mathcal{R}$ be (possibly infinite) TRSs. Let $(\gtrsim, >)$ be a $\mathcal{C}_{\mathcal{E}}$-compatible reduction pair and let π be an argument filtering. If $\mathcal{R}$ is finitely branching, $\mathcal{P} = \mathcal{P}_{\gtrsim^\pi} \cup \mathcal{P}_{>^\pi}$, and $\mathcal{U}_\pi(\mathcal{P}, \mathcal{R}) \subseteq {\gtrsim^\pi}$ then the DP processor $(\mathcal{P}, \mathcal{R}) \mapsto \{(\mathcal{P} \backslash \mathcal{P}_{>^\pi}, \mathcal{R})\}$ is sound.*

Proof sketch. In [GTSKF06] the above theorem is stated and proved for finite TRSs $\mathcal{P}$ and $\mathcal{R}$. A careful inspection of the proof in [GTSKF06] as well as the proofs of related statements in [HM07, Urb04] reveals that it is sufficient that $\mathcal{R}$ is finitely branching. The reason is that then the sets $\{t \mid s \rightarrow_{\mathcal{R}}^{*} t\}$ of reducts of terminating terms s are still guaranteed to be finite. $\square$

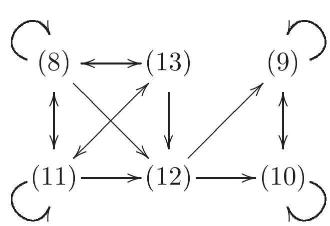
3.2.2 SAT Encoding

We start this section by giving an outline of the main steps of our termination proving procedure. Afterwards we explain which parts are encoded in SAT and how this is actually achieved.

1. First the dependency pairs of $\mathcal{R}$ are computed. Then the strongly connected components (SCCs) in an over-approximation of the dependency graph of $\mathcal{R}$ are determined.
2. In the next step the subterm criterion [HM07] is applied in connection with the recursive SCC algorithm [HM05]. The purpose of this step is to quickly remove components that can be solved in an easier way.
3. At this point we deal with a number of problems of the form $(\mathcal{P}, \mathcal{R})$ where $\mathcal{P}$ is a set of dependency pairs of the original TRS $\mathcal{R}$. Both $\mathcal{P}$ and $\mathcal{R}$ are finite systems over a finite signature. Each of these problems is subjected to the following steps.
4. Predictive labeling (Theorem 3.14) transforms $(\mathcal{P}, \mathcal{R})$ into $(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})$. This new problem generally consists of infinite systems over infinite signatures.
5. Next we apply the reduction pair processor with argument filtering (Theorem 3.15). In our implementation this step is mainly specialized by taking LPO, however we also try polynomial and matrix interpretations (see Sections 1.3.3 and 2.1, respectively), as the underlying reduction order. In order to make progress, there must be at least one rule in $\mathcal{P}$ with the property that all its labeled versions in $\mathcal{P}_{\mathsf{lab}}$ are strictly decreasing.
6. In the next step we return to the problem $(\mathcal{P}, \mathcal{R})$ and remove those rules from $\mathcal{P}$ that were identified in the preceding step. Then we repeat the algorithm on the resulting problem from step 2 onward.

We illustrate this procedure on an example.

Example 3.16. *Consider the TRS $\mathcal{R}$ from Example 1.42 and its dependency pairs $\mathcal{P}$, as presented in Example 1.46. The estimated dependency graph*



has two SCCs: $\{(8), (11), (13)\}$ *and* $\{(9), (10)\}$. *The former is taken care of by two applications of the subterm criterion, first with projection* $\pi(\mathrm{ack}^\sharp) = 1$ *and then with* $\pi(\mathrm{ack}^\sharp) = 2$. *So in step 3 the problem* $(\mathcal{P}, \mathcal{R})$ *with* $\mathcal{P} = \{(9), (10)\}$ *and* $\mathcal{R} = \{(1), \ldots, (7)\}$ *remains. We will label function symbol* $\mathrm{plus}^\sharp$*, so* $\mathcal{G}_\ell(\mathcal{P} \cup \mathcal{R}) = \{\mathrm{s}\}$ *and thus* $\mathcal{U}_\ell(\mathcal{R}) = \varnothing$. *Taking the successor function as semantics for* s *together with the labeling function* $\ell_{\mathrm{plus}^\sharp}(x, y) = x + y$ *produces in step 4 the DP problem* $(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})$ *with* $\mathcal{P}_{\mathsf{lab}}$ *consisting of the rules*

$$\mathrm{plus}^\sharp_{i+j+2}(\mathrm{s}(\mathrm{s}(x)), y) \to \mathrm{plus}^\sharp_{i+j+1}(x, \mathrm{s}(y))$$
$$\mathrm{plus}^\sharp_{i+j+2}(x, \mathrm{s}(\mathrm{s}(y))) \to \mathrm{plus}^\sharp_{i+j+1}(\mathrm{s}(x), y)$$

for all $i, j \geqslant 0$, $\mathcal{R}_{\mathsf{lab}} = \mathcal{R}$, *and* $\mathcal{D}\mathrm{ec} = \{\mathrm{plus}^\sharp_i(x, y) \to \mathrm{plus}^\sharp_j(x, y) \mid i > j \geqslant 0\}$. *The DP problem* $(\mathcal{P}_{\mathsf{lab}}, \mathcal{R}_{\mathsf{lab}} \cup \mathcal{D}\mathrm{ec})$ *is taken care of in step 5 by the argument filtering* $\pi(\mathrm{plus}^\sharp_i) = []$ *for all* i *in combination with the well-founded* LPO *precedence* $\mathrm{plus}^\sharp_i \succ \mathrm{plus}^\sharp_j$ *whenever* $i > j$. *Note that the rules in* $\mathcal{R}_{\mathsf{lab}}$ *are ignored as they are not usable. Since all rules in* $\mathcal{P}_{\mathsf{lab}}$ *are strictly decreasing, there is nothing left to do in step 6.* $\lhd$

We use SAT for steps 4 and 5 of our algorithm. The starting point of our encoding is the approach to semantic labeling with natural numbers and LPO from [KZ06], see Section 3.1, and the encoding of specific instances of Theorem 1.54 in [CSKL$^+$06, ZHM07]. The main challenges are the encoding of

- the search for interpretations f_A and labeling functions ℓ_f,
- the choice of function symbols to be labeled and the corresponding computation of usable rules,
- the induced quasi-model constraints,
- the precedence constraints over the infinite signature of the labeled system, and
- the finite branching condition in Theorem 3.15.

To address the first problem we adapt the SAT encoding of matrix interpretations from [EWZ08].

For the second problem we introduce a new propositional variable L_f for every function symbol f that will indicate whether $L_f \neq \varnothing$. Given those variables we need to compute the set of usable rules for predictive labeling according to Definition 1.41. To this end we introduce propositional variables U_f for every defined symbol f of $\mathcal{R}$ indicating whether the rewrite rules defining f are usable. For a DP problem $(\mathcal{P}, \mathcal{R})$ the encoding of usable rules is expressed as:

$$\omega_{\mathrm{UR}}(\mathcal{P}, \mathcal{R}) = \bigwedge_{f \in \mathcal{F}^\sharp} \Big(\mathsf{L}_f \implies \bigwedge_{g \in \Delta_f(\mathcal{P},\mathcal{R})^*} \mathsf{U}_g\Big)$$

where

$$\Delta_f(\mathcal{P}, \mathcal{R}) = \bigcup_{l \to r \in \mathcal{P} \cup \mathcal{R}} \{g \in \mathcal{F}\mathrm{un}(t) \mid \mathrm{root}(t) = f \text{ and } t \trianglelefteq l \text{ or } t \trianglelefteq r\}.$$

Now the encoding of quasi-model constraints can easily be expressed as

$$\omega_{\mathrm{QM}}(\mathcal{R}) = \bigwedge_{f \in \mathcal{D}_\mathcal{R}} \Big(\mathsf{U}_f \implies \bigwedge_{l \to r \in \mathcal{R}_f} \ulcorner [l]_\mathcal{A} \gtrsim_\mathcal{A} [r]_\mathcal{A} \urcorner\Big).$$

Here $\ulcorner \ldots \urcorner$ converts inequalities into formulas. For that we need to be able to compute term interpretations in the algebra $\mathcal{A}$ and compare them by $\gtrsim_\mathcal{A}$. To that end we can use any technique following the weakly monotone algebra approach from Section 1.3.2, like polynomial interpretations [Lan79] or matrix interpretations [EWZ08] (see Section 2.1). In our implementation we use matrix interpretations with dimensions 1 (which correspond to linear polynomial interpretations) and 2. The reader is referred to [EWZ08] for details on how the corresponding constraints can be encoded. We note that both polynomial and matrix interpretations give $\sqcup$-algebras, which is required for the soundness of predictive labeling (Theorem 3.14).

Note that we encode quasi-model constraints for $\mathcal{R}$ but not for $\mathcal{P}$. The reason is that every DP problem $(\mathcal{P}, \mathcal{R})$ encountered during the execution of our algorithm has the property that the root symbols of the left- and right-hand sides of rules in $\mathcal{P}$ are dependency pair symbols, which do not occur elsewhere in the problem. Hence they are not part of $\mathcal{G}_\ell(\mathcal{P} \cup \mathcal{R})$ and consequently no rule from $\mathcal{P}$ is usable.

The next question is how to restrict the spectrum of possible precedence relations for infinite labeled TRSs in such a way that they have finite representation, can be searched for easily, and ensuring their well-foundedness is feasible. We answer this question in the following definition:

Definition 3.17. [Precedence] For every $f \in \mathcal{F}$ introduce two natural numbers: f_L, the *level* of f and f_SL, the *sublevel* of f. This assignment induces a precedence $\succ_{\mathcal{F}_\mathsf{lab}}$ as follows:

$$f_i \succ_{\mathcal{F}_\mathsf{lab}} g_j \iff f_\mathsf{L} > g_\mathsf{L} \vee \big(f_\mathsf{L} = g_\mathsf{L} \wedge i > j\big) \vee \big(f_\mathsf{L} = g_\mathsf{L} \wedge i \gtrsim j \wedge f_\mathsf{SL} > g_\mathsf{SL}\big)$$

Note that $\succ_{\mathcal{F}_\mathsf{lab}}$ is well-founded since it is obtained as the lexicographic comparison of three well-founded orders. ◇

The straightforward encoding of the (strict) precedence comparisons is presented below. For the computation of labels and their comparison with $>_{\mathcal{A}}$ and $\gtrsim_{\mathcal{A}}$ we may use any approach following the weakly monotone algebra framework.

$$\begin{aligned}\ulcorner f_i \succ_{\mathcal{F}_{\mathsf{lab}}} g_j \urcorner \;&=\; \ulcorner f_{\mathrm{L}} >_{\mathbb{N}} g_{\mathrm{L}} \urcorner \vee \Big(\ulcorner f_{\mathrm{L}} =_{\mathbb{N}} g_{\mathrm{L}} \urcorner \wedge \mathsf{L}_f \wedge \mathsf{L}_g \wedge \big(\ulcorner i >_{\mathcal{A}} j \urcorner \vee \\ &\qquad\qquad (\ulcorner i \gtrsim_{\mathcal{A}} j \urcorner \wedge \ulcorner f_{\mathrm{SL}} >_{\mathbb{N}} g_{\mathrm{SL}} \urcorner)\big)\Big) \\ \ulcorner f_i \succ_{\mathcal{F}_{\mathsf{lab}}} f_j \urcorner \;&=\; \mathsf{L}_f \wedge \ulcorner i >_{\mathcal{A}} j \urcorner\end{aligned}$$

The use of sublevels allows us to represent more precedences on the signatures of infinite labeled TRSs, which increases the termination proving power with only a small reduction in efficiency.

A natural question is how this setting for precedences compares to the one from Section 3.1. It is easy to observe that every precedence from Definition 3.17 has a counterpart in the setting from Definition 3.4. The converse is also true. More precisely, for every well-founded precedence $\succ_{\mathcal{F}_{\mathsf{lab}}}$ obtained from some precedence description, see Definition 3.4, there exists a precedence $\succ'_{\mathcal{F}_{\mathsf{lab}}}$ induced by the level and sublevel mapping from Definition 3.17 (well-founded by definition), such that $\succ_{\mathcal{F}_{\mathsf{lab}}} \subseteq \succ'_{\mathcal{F}_{\mathsf{lab}}}$. So the expressive power of the two approaches is the same.

The last challenge that we address is the requirement of Theorem 3.15 that the TRS $\mathcal{R}_{\mathsf{lab}}$ is finitely branching. For the type of infinite but well structured, parameterized TRSs obtained by labeling finite TRSs this is easy to check. The only source of violation may be a parameterized (labeled) rule where a single labeled instance of a left-hand sides has infinitely many corresponding labeled right-hand sides. In the case of weakly monotone polynomial interpretations this means that a variable must be present in some labels in the right-hand side but not in any label in the corresponding left-hand side. So we define

$$\omega_{\mathrm{FB}}(\mathcal{R}) = \bigwedge_{l \to r \in \mathcal{R}} \; \bigwedge_{x \in \mathcal{V}\mathrm{ar}(r)} \big(\Phi_x(r) \implies \Phi_x(l) \big)$$

with

$$\Phi_x(t) = \bigvee_{f(t_1,\ldots,t_n) \trianglelefteq t} \mathsf{L}_f \wedge a > 0$$

Here a is the coefficient of x when (symbolically) computing the label of f in $f(t_1,\ldots,t_n)$. So $\Phi_x(t)$ evaluates to true when the variable x occurs in some label in term t. Thus $\omega_{\mathrm{FB}}(\mathcal{R})$ expresses that if the value of x is used for obtaining the label of a function symbol occurring in r then this must also be true for a function symbol occurring in l, for every rule $l \to r \in \mathcal{R}$ and every variable x occurring in r.

When using matrix interpretations instead of polynomial interpretations we obtain a similar formula $\omega_{\mathrm{FB}}(\mathcal{R})$, only now variables are interpreted as finite vectors of natural numbers. So in addition to x we must also propagate the position in the vector on which the label depends. We omit the straightforward details.

Combining all ingredients gives us now the final formula for executing steps 4 and 5 of our termination procedure simultaneously:

$$\omega_{\mathrm{UR}}(\mathcal{P},\mathcal{R}) \wedge \omega_{\mathrm{QM}}(\mathcal{R}) \wedge \omega_{\mathrm{FB}}(\mathcal{R}) \wedge \omega_{\mathrm{LAB}}(\mathcal{P},\mathcal{R}) \wedge \omega_{\mathrm{LPO}}(\mathcal{P},\mathcal{R})$$

Some clarifying remarks are in order.

The subformula $\omega_{\mathrm{LAB}}(\mathcal{P},\mathcal{R})$ takes care of computing the labels for all occurrences of all function symbols. (Since the choice of which symbols will be actually labeled is left to the SAT solver, the calculation of labels needs to be encoded for *all* symbols.) This is very similar to the encoding of the algebra computations in $\omega_{\mathrm{QM}}(\mathcal{R})$ and actually we can share most of the code.

The subformula $\omega_{\mathrm{LPO}}(\mathcal{P},\mathcal{R})$ is the encoding of the specialization of Theorem 3.14 to LPO. We adopt the encoding given in [CSKL$^+$06] but since we deal with infinite systems we use as basic building blocks the precedence comparisons sketched on page 80. We compute usable rules with respect to original (unlabeled) system and assume that all labeled versions of a usable unlabeled rule are usable. This gives a correct over-approximation of the usable rules of the labeled TRS.

One thing that seems to be missing in the above formula is the treatment of the rules in $\mathcal{D}$ec. Indeed they are not part of the formula in any way and that is because in the present setting they can be ignored. Regardless of the argument filtering and the precedence (within the constraints of the level/sublevel encoding), the rules in $\mathcal{D}$ec are all (weakly) oriented, do not contribute to the computation of usable rules, and cannot make the system infinitely branching.

The above formula is given to a SAT solver. Three things can happen as a result:

- the SAT solver returns a satisfying assignment, which is translated back to obtain concrete parameters required to execute steps 4 and 5 of our algorithm, or
- the SAT solver returns "unsatisfiable", in which case we know that our approach is not applicable, or
- the SAT solver runs out of time or other resources, in which case we give up without being able to conclude anything.

3.2.3 Experimental Results

We implemented the technique described in the preceding section in the termination prover TPA 2007 (see Chapter 4 for introduction to the prover), using the MiniSat SAT solver [ES03]. In this section we evaluate our method on a number of examples from the TPDB [TPD]). All experiments involving TPA were performed

technique	tool	yes	time	timeout
matrix interpretations	Jambox	505	N/A	N/A
	TPA 2007	498	3541	30
LPO	AProVE	380	193	0
	TPA 2007	372	191	0

Table 3.1 Comparison to other tools.

on a machine equipped with an Intel® Pentium® 4, 3.00 GHz processor. Experimental data for other termination tools are taken from the respective publications (due to the difficulty of obtaining those tools in the configuration required for our experiments).

A very natural benchmark for our approach would be the comparison with results from Section 3.1. Unfortunately the substantial difference in the approach to semantic labeling with natural numbers makes any decent comparison difficult. We are convinced however that the direct approach from Section 3.1 would be absolutely infeasible for exploring the much larger search space resulting from using arbitrary interpretations with bounded coefficients instead of only a small number of predefined interpretations.

We begin by evaluating two basic ingredients of our implementation, matrix interpretations and LPO with argument filtering (both without semantic or predictive labeling), against reference implementations: in Jambox [EWZ08] for the former and in AProVE [CSKL+06] for the latter. Both implementations use more or less the same setup: dependency pairs with usable rules and subterm criterion. The results in Table 3.1 are based on TPDB ver. 2.0, more precisely on the 773 TRSs from the termination category of this database. We use on an older version of the database for this experiment as we cannot obtain those tools in a required configuration and hence we need to rely on figures published in [EWZ08] and [CSKL+06].

The columns "yes", "time", and "touts" indicate the number of successful termination proofs, the total time (in seconds) spent on the TRSs in the problem set and the number of timeouts that occurred. We used a 60 seconds time limit.

The experiments for matrix interpretations use 2×2 matrices, 2 bits for the matrix entries, and 3 bits to represent the values of intermediate results. The slightly lower score of TPA compared to Jambox is probably due to a more sophisticated approximation of the dependency graph in the latter. No timing information is given in [EWZ08] but the authors write "[...] we took the time limit of 1 minute [...] this time was hardly ever consumed [...] average computation time for all proofs is around 1 second". This suggests that our implementation is far from optimal. Indeed, we did not invest much time in optimizing the encoding. This seems to be a good starting point for improving the results for predictive labeling

	technique	60 seconds timeout yes	time	touts	10 minutes timeout yes	time	touts
1×1	SL	409	3063	22	415	5325	2
	PL	428	2927	16	433	5166	2
	PL′	397	2234	15	401	3858	2
2×2	SL	464	8960	70	477	26194	23
	PL	495	8985	68	506	26293	23
	PL′	489	6113	44	497	16022	11

Table 3.2 Experiments with TPA on TPDB version 4.0.

presented below.

The slightly higher score of AProVE in the LPO experiments is likely due to a different graph approximation algorithm. The execution speeds are almost the same but one needs to keep in mind that the results were obtained on different machines and hence cannot be compared directly.

Table 3.2 summarizes the experiments performed with TPA on the 975 TRSs from TPDB ver. 4.0. All experiments were performed with time limits of 60 seconds and 10 minutes. The first group of results is based on semantic/predictive labeling with matrix interpretations of dimension 1 (equivalent to linear polynomials) used for both interpretations and labels:

- SL means semantic labeling (Theorem 3.13) where all symbols are labeled and all rules are considered for the quasi-model requirement,
- PL stands for predictive labeling (Theorem 3.14) and corresponds to the approach described in the preceding sections,
- PL′ is a variant with a simple heuristic for the choice of labeled symbols; instead of leaving this decision to SAT all dependency pair symbols are labeled and only them.

A first observation is that predictive labeling is more powerful than semantic labeling—it proves termination of an additional 19 TRSs — without almost any difference in execution speed. The heuristic brings a considerable speedup at the expense of termination proving power.

For the second group of results we use the same methods as for the first group but now 2×2 matrices are used for interpretations and labels. Again predictive labeling is more powerful than semantic labeling, without being slower. It is interesting to observe that the price in termination proving power of the heuristic is much less than for dimension 1. This can be intuitively explained by the more powerful algebraic structure used for the labeling functions, which makes it possible to put more information in the labels and hence counter the reduced flexibility in the choice of function symbols to label. A similar line of reasoning could lead to the belief that in this case it is also easier to satisfy quasi-model constraints and thereby diminishing the improvement of predictive labeling but the difference of 31 TRSs between SL and PL proves this hypothesis wrong. It is worth noting that even for the slowest variant (PL with matrices of dimension 2×2) the average time for successful proof is around 4 seconds.

3.3 Conclusions

In this chapter we presented a direct way of automating semantic labeling with natural numbers. For the semantics we used a small fixed set of functions over natural numbers — few polynomial functions extended with min and max for binary symbols. After applying semantic labeling, for the infinite, labeled systems termination proofs are obtained using RPO. This technique was implemented in the 2005 version of TPA and allowed to solved the challenging SUBST TRS.

Later on we presented a new, improved approach extending the theory of predictive labeling to a dependency pairs setting. We also presented the ideas behind the SAT based implementation of this technique in the 2007 version of TPA. Experimental results confirm the feasibility of this technique. The advantages over the old approach can be summarized as:

- the quasi-model restriction is relaxed by using predictive labeling,
- the integration with dependency pairs makes the approach more powerful,
- the SAT encoding enables the use of unrestricted polynomial and matrix interpretations for function symbols, instead of the, previously used, small, fixed set of functions.

The only disadvantage compared with the old approach is that we fail to cover the possibility of using min and max in interpretations. This is essential, for instance to deal with the SUBST TRS; hence this new approach fails for this system. However, there is a recent work [FGM$^+$08] on extending polynomial interpretations with max, in a systematical way (our treatment of those functions was rather ad hoc). Those new results should be applicable in the setting of predictive labeling and incorporating them to this method and evaluating the results is our future work.

An important theoretical question is whether the finite branching condition in Theorem 3.15 is essential. Disabling the $\omega_{\mathrm{FB}}(\mathcal{R})$ conjunct in our encoding allows to "prove" the termination of two more TRSs from the TPDB. One of these TRSs is presented below.

Example 3.18. *Consider the following TRS (Thiemann/factorial1.trs) computing the factorial function:*

$$
\begin{aligned}
\mathrm{plus}(0,x) &\to x & \mathrm{plus}(\mathrm{s}(x),y) &\to \mathrm{s}(\mathrm{plus}(\mathrm{p}(\mathrm{s}(x)),y)) \\
\mathrm{times}(0,y) &\to 0 & \mathrm{times}(\mathrm{s}(x),y) &\to \mathrm{plus}(y,\mathrm{times}(\mathrm{p}(\mathrm{s}(x)),y)) \\
\mathrm{p}(\mathrm{s}(0)) &\to 0 & \mathrm{p}(\mathrm{s}(\mathrm{s}(x))) &\to \mathrm{s}(\mathrm{p}(\mathrm{s}(x))) \\
\mathrm{fac}(0,x) &\to x & \mathrm{fac}(\mathrm{s}(x),y) &\to \mathrm{fac}(\mathrm{p}(\mathrm{s}(x)),\mathrm{times}(\mathrm{s}(x),y)) \\
\mathrm{factorial}(x) &\to \mathrm{fac}(x,s(0))
\end{aligned}
$$

There are ten dependency pairs and the estimated dependency graph contains four single node SCCs. Only one of them, consisting of the dependency pair

$$\mathrm{fac}^{\sharp}(\mathrm{s}(x),y) \to \mathrm{fac}^{\sharp}(\mathrm{p}(\mathrm{s}(x)),\mathrm{times}(\mathrm{s}(x),y))$$

is problematic. It could be solved using matrices of dimension 2 *by labeling* s *and* p*, but it is essential that the interpretation of* plus *depends on its second argument. Then the label of the root symbol of the right-hand side of the rule* $\mathrm{plus}(\mathrm{s}(x),y) \to \mathrm{s}(\mathrm{plus}(\mathrm{p}(\mathrm{s}(x)),y))$ *depends on the assignment to* y *whereas in the left-hand side there is only one labeled* s *symbol with* x *as its argument so its label is necessarily independent of the value of* y*. This makes* $\mathcal{R}_{\mathsf{lab}}$ *non-finitely branching and hence the termination proof is out of reach with our approach.* ◁

In the proof sketch of Theorem 3.15 we remarked that it relies on the finite branching condition. The key idea in the proof goes back to a modularity result for termination of Gramlich [Gra94], in which the same finite branching condition is required. By using a much more complicated construction, Ohlebusch [Ohl94] showed that the finite branching condition in the modularity result is not essential. It is worthwhile to investigate whether the proof technique in [Ohl94] can be used to generalize Theorem 3.15.

Chapter 4

(T)TPA: (Trusted) Termination Proved Automatically

In recent years the research on methods for proving termination focused on the automation of this process. Tools are developed to automatically (dis)prove termination of TRSs and their efficiency is tested in an annual termination competition.

Termination methods usually involve a large, often infinite, search space. Restricting this space to make the search feasible, without loosing too much proving power, and then efficiently exploring it, is a big challenge. Recently, reduction to SAT problems and employment of efficient SAT solvers is often a method of choice.

Modern termination provers rely not on one but on a (sometimes large) number of termination techniques. Combining them in an effective manner, to maximize proving power while remaining efficient, is yet another challenge.

Finally, with this growing complexity ensuring reliability of the tools is not an easy task. In this thesis we try to address one way of handling this problem: certification of the results produced by termination provers with the use of a formal proof assistant/checker. We have introduced this concept in Section 1.4.

In this chapter we will present the termination prover TPA. It started as a prototype to investigate feasibility of using semantic labeling over infinite models for automatically proving termination. Later it grew into a tool of its own, participat-

---

An earlier version of a part of this chapter appeared as: A. Koprowski, TPA: Termination Proved Automatically, In F. Pfenning ed., *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, Seattle, WA, USA, volume 4098 of Lecture Notes in Computer Science, pp. 257–266, Springer-Verlag, August 2006.

ing regularly in the termination competition. Recently its focus shifted towards certified proofs and in 2007 it won the first edition of the certified competition.

We will give an overview of TPA and its evaluation in Section 4.1 and we will present some ideas for future development of the tool in Section 4.2.

4.1 TPA — Termination Proved Automatically

4.1.1 Motivation

Before the development of TPA started, a number of tools for proving termination automatically was available, hence it is natural to ask why creating another one. There are three main reasons why TPA has been developed and they are listed below in no particular order.

- **Semantic labeling with natural numbers**
  Semantic labeling was introduced in Section 1.3.5. Its variant with the model over two or three element sets was used in some tools preceding TPA. However the infinite model variants were so far considered not to be suitable for automation. We believed that this could be accomplished and the theoretical framework for automation of semantic labeling with natural numbers, and, later, predictive labeling, was presented in Chapter 3. The need to evaluate those ideas in practice was the driving force for the first prototype of TPA.

- **Relative termination**
  Relative termination was introduced in Section 1.3.1. Prior to TPA there was no support for relative termination in tools for proving termination of TRSs. However, it is a natural concept occurring in practice. In Chapter 5 we will present the extension of the work of Giesl and Zantema on methods for verification of liveness properties using rewriting techniques [GZ03a]. Extending their framework with the notion of fairness naturally leads to the notion of relative termination. Lucas and Meseguer conducted research on termination of concurrent systems under fairness assumptions [LM08]. In their setting they again use the notion of relative termination to establish the property of fair-termination. They use TPA for proving relative termination.

- **Certified termination**
  We introduced the CoLoR project and the idea of certification of termination proofs in Section 1.4. The success of this project requires cooperation between the tool authors and CoLoR developers. While being involved in the CoLoR project, also having our own tool was helpful in bridging the gap between those two communities and propagating the idea of formal verification of termination proofs. Indeed this idea has been recognized by the community. In 2007 the termination competition was extended with a special

category of certified termination and in 2007 and 2008 the first two editions of the "Workshop on Certified Termination" were organized [WSc].

4.1.2 Overview and Experimental Results

TPA is implemented in Objective Caml (OCaml) [OCA], making its byte code version available for all major platforms (with a native version for Linux). It is equipped with command line interface. The first version of the tool was developed in 2005 and supported the following methods for proving termination:

- polynomial interpretations [Lan79] (Section 1.3.3),
- recursive path order [Der82] (Section 1.3.4),
- semantic labeling [Zan95] (Section 1.3.5),
- dependency pairs [AG00] (Section 1.3.7),
- dummy elimination [FZ95] and
- reduction of right hand sides [Zan05].

It was first developed to evaluate feasibility of the technique of semantic labeling with natural numbers, presented in Section 3.1. Then, a few more techniques were added and TPA took part in the termination competition in 2005. In the standard rewriting category it was third, out of the six participating tools, solving 407 out of the 773 termination problems. The winner, AProVE, produced 576 successful proofs and TTT was second with the score of 509.

After some minor modifications TPA took part in the termination competition of 2006 and again took the third place (with six participating tools) solving 422 out of the 865 problems, coming after AProVE (638) and Jambox (626). Using the method of semantic labeling with natural numbers it was the only tool capable of proving termination of the SUBST TRS (see Example 3.1). The automatic proof produced by TPA (after minor modifications for presentation purposes) is presented in Figure 4.1.

All the aforementioned methods were implemented in TPA using some direct search procedures (involving exhaustive inspection of some finite part of the search space). After the competition of 2006 many of those techniques were instead re-implemented using an encoding into SAT, in accordance with the prevailing trend in this area. Moreover the tool was extended with the following termination techniques:

- matrix interpretations [EWZ08] (Section 2.1),

```
  TPA v.1.1
Result: TRS is terminating

Default interpretations for symbols are not printed. For polynomial
interpretations and semantic labelling over N\{0,1} defaults are 2
for constants, identity for unary symbols and x+y-2 for binary symbols.
For semantic labelling over {0,1} (booleans) defaults are 0 for constants,
identity for unary symbols and disjunction for binary symbols.

[1] TRS as loaded from the input file:
(1) a(lambda(x),y) -> lambda(a(x,p(1,a(y,t))))  (5) a(x,y) -> x
(2) a(p(x,y),z) -> p(a(x,z),a(y,z))             (6) a(x,y) -> y
(3) a(a(x,y),z) -> a(x,a(y,z))                  (7) p(x,y) -> x
(4) lambda(x) -> x                              (8) p(x,y) -> y

[2]  Label this TRS using following interpretation over N\{0,1}:
  [lambda(x)] = x + 1,   [p(x,y)] = max(x, y),  rest default

  This interpretation is a quasi-model and yields following TRS:
(1 ) a{i + 1,j}(lambda{i}(x),y) ->
        lambda{j + i - 2}(a{i,j}(x,p{2,j}(1,a{j,2}(y,t))))
(2<) a{i,k}(p{i,j}(x,y),z) ->
        p{k + i - 2,k + j - 2}(a{i,k}(x,z),a{j,k}(y,z)) for i >= j
(2>) a{j,k}(p{i,j}(x,y),z) ->
        p{k + i - 2,k + j - 2}(a{i,k}(x,z),a{j,k}(y,z)) for j >= i
(3 ) a{j + i - 2,k}(a{i,j}(x,y),z) -> a{i,k + j - 2}(x,a{j,k}(y,z))
(4 ) lambda{i}(x) -> x
(5 ) a{i,j}(x,y) -> x
(6 ) a{i,j}(x,y) -> y
(7<) p{i,j}(x,y) -> x for i >= j      (8<) p{i,j}(x,y) -> y for i >= j
(7>) p{i,j}(x,y) -> x for j >= i      (8>) p{i,j}(x,y) -> y for j >= i
(D1) lambda{i + 1}(x) ->= lambda{i}(x)
(D2) a{i + 1,j}(x,y) ->= a{i,j}(x,y)  (D4) p{i + 1,j}(x,y) ->= p{i,j}(x,y)
(D3) a{i,j + 1}(x,y) ->= a{i,j}(x,y)  (D5) p{i,j + 1}(x,y) ->= p{i,j}(x,y)

[3]  Using RPO with the following precedence:
Status:
 a: lexicographic, left-to-right
Precedence:
 a_{i,j} >/=/< a_{k,l} if i + j >/=/< k + l
 a_{i,j} > lambda_{k}
 a_{i,j} > p_{k,l}
 a_{i,j} > 1
 a_{i,j} > t
 lambda_{i} >/=/< lambda_{k} if i >/=/< k
 p_{i,j} >/=/< p_{k,l} if i + j >/=/< k + l
  the following rules can be oriented and removed:
(D1)-(D5), (4)-(6), (1), (2<), (2>), (3), (7<), (7>), (8<), (8>)
```

Figure 4.1 The termination proof of the SUBST system produced by TPA in the termination competition of 2006

- argument filtering [AG00] (Section 1.3.7),
- predictive labeling in the dependency pair setting [KM07] (Section 3.2).

TPA was supposed to take part in the 2007 competition equipped only with the technique of predictive labeling, as presented in Section 3.2. Unfortunately, due to some last minute changes, a flaw was introduced and the tool had to be withdrawn from the contest.

In 2007 the focus of TPA shifted towards certified proofs (introduced in Section 1.4). That year the new category of certified termination was introduced in the competition. The combined entry of TPA, as the termination prover, and CoLoR, as the certification back-end, was the winning entry in this new category. It achieved the score of 354, meaning that for 354 out of the total 975 TRSs used in the competition, TPA could find a termination proof and, using CoLoR, correctness of this proof could be verified by Coq.

The techniques used by TPA were constrained by the techniques available in CoLoR and Rainbow at that time and consisted of:

- the basic theorem of dependency pairs (Theorem 1.49),
- polynomial interpretations (Section 2.3.3) and
- matrix interpretations (Section 2.1 and Section 2.3.4).

Arctic interpretations (Section 2.3.3) were not yet available at that time.

Due to the fact that this category was introduced only that year there were only two other participants. The termination prover C*i*ME [CMMU] using the Coccinelle [CCF$^+$07] library to certify termination results got the second place with a score of 317. The third participating tool was the entry of $\mathsf{T_TT_2}$ [KSZM] again using CoLoR as the certifying back-end with a score of 289.

In Table 4.1 we present performance of all the versions of TPA mentioned above, evaluated on the same TPDB 4.0 used in the termination competition of 2007 and consisting of 975 TRSs. For completeness we also include the TPA 2007 version with predictive labeling (without the aforementioned flaw), even though in the end it did not participate in the competition. The second part of the table presents performance of the version of TPA capable of producing verifiable proofs, which participated in the certified competition of 2007. The first row, "TPA 2007 (cert.)", shows performance for proof search performed by TPA and the second one, "+ CoLoR", the performance of CoLoR verification applied to the set of proofs found in the first step. Identical timeouts were used, separately, for those two steps.

All the runs were performed on an Intel® Pentium® 4, 3.00 GHz machine with a 60 seconds and 10 minutes timeouts. The columns "yes", "time", and "touts" indicate the number of successful termination proofs, the total time (in seconds) spent on the TRSs in the problem set and the number of timeouts that occurred.

	60 seconds timeout			10 minutes timeout		
tool version	yes	time	touts	yes	time	touts
TPA 2005	366	34302	558	379	329096	538
TPA 2006	370	29306	230	383	39235	8
TPA 2007	495	8985	68	506	26293	23
TPA 2007 (cert.)	354	32416	472	361	190588	258
+ CoLoR	352	2383	2	359	3454	2

Table 4.1 Performance of successive versions of TPA on TPDB ver. 4.0.

4.2 TTPA — Trusted Termination Proved Automatically

TTPA, for *Trusted Termination Proved Automatically*, is the successor of TPA. It tries to address two challenges that we think are very important in the area of proving termination of term rewriting, namely:

1. reliability of the produced results, which is becoming a problem due to the complexity of the tools and their proofs,
2. a very high effort required to build a new tool or even a prototype, as a (successful) tool needs to have a large number of techniques at its disposal and one typically is not interested in re-implementing those.

Those two challenges will be, to some extend, addressed by TTPA by means of:

1. *Changing focus to certified proofs.* This change in philosophy is reflected in the new name of the tool indicating its main objective: producing termination proofs certifiable with CoLoR. For that the primary focus of the tool will be on the termination methods formalized and supported in CoLoR.
2. TTPA is written from scratch. One of the goals in the development process is to produce a clean, well documented code. *The tool is open-source* which will hopefully make it possible for others to benefit from, or even reuse, parts of its code. This, in combination with the first point, will hopefully enable it to serve as a "reference implementation" for certifiable termination provers.

The development of the tool is work in progress.

Chapter 5

Application: Proving Liveness Properties

Usually, *liveness* is roughly defined as: "*something will eventually happen*" and it is often remarked that "*termination is a particular case of liveness*". In [GZ03a] the relationship between liveness and termination was investigated in more detail, and it was observed that conversely liveness can be seen as termination of a modified relation. Since various techniques have been developed to prove termination automatically, some of them discussed in previous chapters, an obvious goal is to apply these techniques in order to prove liveness properties. In [GZ03a] a method for transforming a class of liveness problems to problems of termination of TRSs has been proposed. For a slightly different setting another approach was proposed in [GZ03b].

In [GZ03a] two transformations were given. The first one, sound and complete, even for extremely simple liveness problems results in complicated TRSs for which proving termination, especially in an automated way, is very difficult. That was the motivation for a, much simpler, transformation, which is sound but not complete.

In this chapter this approach is extended in two ways. First we extend the basic framework to *fair computations*. That means that we do not restrict to the basic notion of liveness stating that any computation eventually reaches a good state, but we do this for fair computations which contain some essential computation steps infinitely often. Fairness has been studied extensively in [Fra86]. In applications, one is often interested in the behavior of fair computations rather than of arbitrary

---

An earlier version of this chapter appeared as: A. Koprowski, H. Zantema, Proving Liveness with Fairness using Rewriting, In B. Gramlich ed., *Proceedings of the 5th International Workshop on Frontiers of Combining Systems (FroCoS '05)*, Vienna, Austria, volume 3717 of Lecture Notes in Computer Science, pp. 232–247, Springer-Verlag, September 2005.

ones. For instance, in a waiting line protocol one may want to prove that eventually all initial clients will be served. If it is allowed that infinitely many new clients come in, one may think of an infinite computation in which this does not hold: infinitely many new clients come in but no client is ever served. However, if serving of clients is defined to be the essential computation step, in a corresponding fair computation it can be proved that eventually all old clients will be served. It turns out that just like liveness corresponds to termination, liveness in fair computations corresponds to relative termination. So combining liveness and fairness is a main contribution of the work presented in this chapter.

The second extension is the following. It turns out that the simple transformation presented in [GZ03a] often results in non-terminating TRSs, and therefore is not applicable, also in liveness problems not involving fairness. Therefore, we propose an alternative transformation. Our transformation is slightly more complicated than the simple transformation from [GZ03a], but much simpler than the sound and complete transformation from [GZ03a]. However, assuming some mild conditions, in this chapter we show that our transformation is sound and complete too. Moreover, we show in two examples that our transformation results in TRSs for which (relative) termination can be proved fully automatically. In particular we consider the classical readers-writers synchronization problem, in which the priority of access is controlled in an obvious way. The desired liveness property states that every process in the system eventually gets access to the resource. Using our technique we succeed in automatically proving this liveness property with the TPA termination prover (see Chapter 4). Both examples involve infinite state spaces and hence the standard model checking techniques are not applicable to them.

This chapter is organized as follows. In Section 5.1 the general framework from [GZ03a] is extended in order to deal with liveness with fairness. Next, in Section 5.2, the new transformation is introduced and the corresponding theorems on soundness and completeness are given. Finally, in Section 5.3, two examples are presented in which this new approach has been applied. We conclude in Section 5.4.

5.1 Liveness with Fairness Conditions

5.1.1 Liveness in Abstract Reduction

First we present the framework as described in [GZ03a] with no more than necessary details to understand its extension given later. For a more elaborate description we refer to the original article.

We give the model of the system that should be verified in the framework of abstract reduction. We assume a set of states $\mathcal{H}$ and a binary relation on states expressing computation steps, $\rightarrow \subseteq \mathcal{H} \times \mathcal{H}$. We call a reduction sequence *maximal* if it is either infinite or its last element is in $\mathrm{NF}(\rightarrow)$.

With respect to a set of initial states $\mathcal{I} \subseteq \mathcal{H}$ and a set of good states $\mathcal{G} \subseteq \mathcal{H}$, we say that the liveness property $\mathrm{Live}(\mathcal{I}, \to, \mathcal{G})$ holds if all maximal $\to$-reductions starting in $\mathcal{I}$ contain an element from $\mathcal{G}$. More precisely:

Definition 5.1. [Liveness] Let $\mathcal{H}$ be a set of states, $\to \subseteq \mathcal{H} \times \mathcal{H}$; $\mathcal{G}, \mathcal{I} \subseteq \mathcal{H}$. Then $\mathrm{Live}(\mathcal{I}, \to, \mathcal{G})$ holds iff

- $\forall_{t_1, t_2, \ldots}\ t_1 \in \mathcal{I} \wedge (\forall_i\ t_i \to t_{i+1}) \implies \exists_i\ t_i \in \mathcal{G}$, and
- $\forall_{t_1, t_2, \ldots, t_n}\ t_1 \in \mathcal{I} \wedge t_n \in \mathrm{NF}(\to) \wedge (\forall_{1 \leqslant i \leqslant n-1}\ t_i \to t_{i+1}) \implies \exists_{1 \leqslant i \leqslant n}\ t_i \in \mathcal{G}$.

Furthermore we define the restricted computation relation

$$\to_{\mathcal{G}} \equiv \{(s, t) \mid s \to t \wedge s \notin \mathcal{G}\} \qquad \diamond$$

The following theorem relates liveness to termination of $\to_{\mathcal{G}}$.

Theorem 5.2 ([GZ03a])**.** *If* $\mathrm{NF}(\mathcal{I}, \to) \subseteq \mathcal{G}$ *then* $\mathrm{Live}(\mathcal{I}, \to, \mathcal{G})$ *iff* $\mathrm{SN}(\mathcal{I}, \to_{\mathcal{G}})$. $\square$

5.1.2 Liveness with Fairness in Abstract Reduction

In liveness we are mainly interested in the behavior of infinite reduction sequences, or shortly, infinite reductions. However, in many applications one is not interested in arbitrary infinite reductions but in infinite fair reductions, defined as follows. Instead of a single rewrite relation $\to$ we have two relations $\to, \tilde{\to} \subseteq \mathcal{H} \times \mathcal{H}$. An infinite reduction in $\to \cup \tilde{\to}$ is called *fair* (with respect to $\to$) if it contains infinitely many $\to$-steps. Finally we say that liveness for fair reductions starting in $\mathcal{I}$ with respect to $\to$, $\tilde{\to}$ and $\mathcal{G}$ holds, denoted as $\mathrm{Live}(\mathcal{I}, \to, \tilde{\to}, \mathcal{G})$, if any fair $\to \cup \tilde{\to}$ reduction starting in $\mathcal{I}$ contains an element of $\mathcal{G}$. Note that all fair reductions are infinite, hence in investigating liveness with fairness we are only interested in systems with infinite behavior.

Definition 5.3. [Liveness with fairness] Let $\mathcal{H}$ be a set of states, $\to, \tilde{\to} \subseteq \mathcal{H} \times \mathcal{H}$; $\mathcal{G}, \mathcal{I} \subseteq \mathcal{H}$. Then liveness for fair reductions with respect to $\mathcal{I}, \to, \tilde{\to}$ and $\mathcal{G}$, $\mathrm{Live}(\mathcal{I}, \to, \tilde{\to}, \mathcal{G})$, holds iff

$$\forall_{t_1, t_2, \ldots}\ t_1 \in \mathcal{I} \wedge (\forall_i\ t_i \to t_{i+1} \vee t_i \tilde{\to} t_{i+1}) \wedge (\forall_i\ \exists_{j > i}\ t_j \to t_{j+1}) \implies \exists_i\ t_i \in \mathcal{G} \quad \diamond$$

Our definition is based on the notion of relative termination (see Section 1.1). The result of the next theorem gives us a method of verifying liveness with fairness requirements.

Theorem 5.4. $\mathrm{Live}(\mathcal{I}, \to, \tilde{\to}, \mathcal{G})$ *holds iff* $\mathrm{SN}(\mathcal{I}, \to_{\mathcal{G}} / \tilde{\to}_{\mathcal{G}})$.

Proof. ($\Rightarrow$) Assume that $\mathrm{Live}(\mathcal{I}, \to, \overset{\sim}{\to}, \mathcal{G})$ holds and $\mathrm{SN}(\mathcal{I}, \to_{\mathcal{G}} \,/\, \overset{\sim}{\to}_{\mathcal{G}})$ does not hold. From the latter we get that there is an infinite, fair reduction sequence $t_1, t_2, \ldots$ with $t_1 \in \mathcal{I}$ and $\forall_i\ t_i \to_{\mathcal{G}} t_{i+1} \vee t_i \overset{\sim}{\to}_{\mathcal{G}} t_{i+1}$. From the definition of $\to_{\mathcal{G}}$ all $t_i \notin \mathcal{G}$. But then this reduction sequence is a counter-example for $\mathrm{Live}(\mathcal{I}, \to, \overset{\sim}{\to}, \mathcal{G})$.

($\Leftarrow$) $\mathrm{SN}(\mathcal{I}, \to_{\mathcal{G}} \,/\, \overset{\sim}{\to}_{\mathcal{G}})$, so in every infinite, fair $\to \cup \overset{\sim}{\to}$ reduction starting in $\mathcal{I}$ there is an element from $\mathcal{G}$ (which blocks further $\to_{\mathcal{G}} \cup \overset{\sim}{\to}_{\mathcal{G}}$ reductions) and that is exactly what the definition of $\mathrm{Live}(\mathcal{I}, \to, \overset{\sim}{\to}, \mathcal{G})$ calls for. ☐

5.1.3 Liveness with Fairness in Term Rewriting

In previous sections we described the transition relation by means of abstract reductions, and related liveness of $\to$ to termination of $\to_{\mathcal{G}}$. Our goal is to employ techniques for proving termination of rewriting in order to prove liveness properties. To that end a transformation is required since usually $\to_{\mathcal{G}}$ is not a rewrite relation even if $\to$ is a rewrite relation.

Now we will represent the computation states by terms, so $\mathcal{H}$ becomes $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\mathcal{I}, \mathcal{G} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$. Abstract reduction relations $\to$ and $\overset{\sim}{\to}$ now correspond to two TRSs over the same signature $\mathcal{F}$: $\mathcal{R}$ and $\mathcal{R}_{\sim}$, respectively. As before we write $\to$ as a shorthand for $\to_{\mathcal{R}}$ and $\overset{\sim}{\to}$ for $\to_{\mathcal{R}_{\sim}}$. Just like it is usual to write $\mathrm{SN}(\mathcal{R})$ rather than $\mathrm{SN}(\to_{\mathcal{R}})$, we will write $\mathrm{Live}(\mathcal{I}, \mathcal{R}, \mathcal{R}_{\sim}, \mathcal{G})$ rather than $\mathrm{Live}(\mathcal{I}, \to_{\mathcal{R}}, \to_{\mathcal{R}_{\sim}}, \mathcal{G})$.

Now, again after [GZ03a], we will introduce the notion of top TRSs, which we are going to use to model liveness problems.

Definition 5.5. [Top TRSs] Let $\mathcal{F}$ be a signature and top be a fresh unary symbol in this signature, that is $\mathrm{top} \notin \mathcal{F}$. A term $t \in \mathcal{T}(\mathcal{F} \cup \{\mathrm{top}\}, \mathcal{V})$ is called a *top term* if it contains exactly one instance of the top symbol, at the root of the term. We denote the set of top terms over $\mathcal{F}$ and $\mathcal{V}$ as $\mathcal{T}_{\mathrm{top}}(\mathcal{F}, \mathcal{V})$.

A TRS over $\mathcal{F} \cup \{\mathrm{top}\}$ is called a *top term rewrite system* (top TRS) if for all its rules $l \to r$ one of the following holds:

- Both l and r are top terms. Then we call this rule a *top rule.*
- Both l and r do not contain an instance of top symbol. Then the rule is called a *non-top rule.* ◇

Clearly for top TRSs every reduction starting in a top term only contains top terms. In the reminder we restrict ourselves to liveness with respect to

- reduction relations described by top TRSs,
- the set of initial states consisting of all top terms, and

- the set of good states of the form:

$$G(P) = \{t \in \mathcal{T}_{\mathrm{top}}(\mathcal{F}, \mathcal{V}) \mid \neg\exists_{p \in P,\ \sigma,\ C}\ t = C[p\sigma]\}$$

  for some set $P \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$, that is $G(P)$ represents top terms not containing an instance of any of the terms from P.

So we are going to investigate liveness properties of the form:

$$\mathrm{Live}(\mathcal{T}_{\mathrm{top}}(\mathcal{F}, \mathcal{V}), \mathcal{R}, \mathcal{R}_{\sim}, G(P))$$

for some top TRSs $\mathcal{R}$ and $\mathcal{R}_{\sim}$. This is equivalent to proving that every infinite fair reduction of top terms contains a term which does not contain an instance of any of the terms from P.

As we will show later this type of question can be transformed to a relative termination question of an ordinary TRS. This allows us to employ the techniques for proving relative termination for TRSs to verify liveness properties. Also, while quite restricted, this setting seems to be general enough to be able to cope with some interesting and practical examples, two of which will be presented at the end of this chapter.

5.2 An Alternative Transformation

5.2.1 Motivation

We are seeking a transformation with the property that relative termination of the transformed pair of systems implies that the liveness property in question holds (even better if we can have equivalence). In [GZ03a] two such transformations were proposed: the first one sound and complete (equivalence between termination and liveness holds) and the second one only sound (termination implies liveness but not the other way around) but significantly simpler. Experiments with them show that the former is so complex that, although it is a nice theoretical result, in practice it leads to TRSs far too complicated for present termination techniques to deal with, especially in an automated way. The sound transformation does not have this disadvantage but in several examples it is not powerful enough, leading to non-terminating TRSs, while the desired liveness property does hold.

In this section we propose an alternative transformation avoiding the aforementioned problems. But before we do that we will shortly introduce the sound transformation LS from [GZ03a] where $P = \{p\}$. As in our presentation also LS can be easily generalized to allow P to contain more than one element, as remarked in [GZ03a].

Definition 5.6. [LS] Let $\mathcal{R}$ be a top TRS over $\mathcal{F} \cup \{\text{top}\}$ and $p \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. We define $\text{LS}(\mathcal{R}, p)$ to consist of the following rules:

$$\begin{aligned}
l &\to r && \text{for all non-top rules } l \to r \text{ in } \mathcal{R} \\
\text{top}(l) &\to \text{top}(\text{check}(r)) && \text{for all top rules top}(l) \to \text{top}(r) \text{ in } \mathcal{R} \\
\text{check}(f(x_1, \ldots, x_n)) &\to f(x_1, \ldots, \text{check}(x_i), \ldots, x_n) && \\
& && \text{for all } f \in \mathcal{F},\ 1 \leqslant i \leqslant \text{arity}(f) \\
\text{check}(p) &\to p && \diamond
\end{aligned}$$

While $\text{LS}(\mathcal{R}, p)$ is sound, it is not complete. This is illustrated in the following example.

Example 5.7. *Consider the TRS consisting of the following two rules:*

$$\text{top}(\text{f}(x, \text{b})) \to \text{top}(\text{f}(\text{b}, \text{b}))$$
$$\text{a} \to \text{b}$$

Normal forms do not contain symbol a *and in every infinite reduction after finitely many steps only term* $\text{top}(\text{f}(\text{b}, \text{b}))$ *occurs, so liveness for* $p = \text{a}$ *holds. However,* $\text{LS}(\mathcal{R}, p)$ *admits an infinite reduction, namely:*

$$\text{top}(\text{check}(\text{f}(\text{b}, \text{b}))) \rightleftarrows \text{top}(\text{f}(\text{check}(\text{b}), \text{b}))$$

The reason why things go wrong here is that the idea of the transformation is that when p does not occur in a term then every application of top rule produces check *symbol that cannot be removed. It can only be propagated downwards in the term and the hope is that this will block further reductions and hence infer termination. But in the above example and in the examples that we will see in Section 5.3, this is not the case, and hence* LS *fails for them.* $\lhd$

5.2.2 Definition of the Transformation

We give a transformation inspired by the sound and complete transformation presented in [GZ03a] but significantly simpler so that obtained systems can still be treated with tools for automatic termination proving. It can deal with a much broader class of liveness problems than the sound transformation from [GZ03a] and, although somehow more complicated, it is still suitable for automatic termination provers. We present it for only one unary top symbol but generalization to more top symbols and/or different arities is straightforward.

Definition 5.8. [LT] Let $\mathcal{R}$ and $\mathcal{R}_\sim$ be top TRSs over $\mathcal{F} \cup \{\text{top}\}$ and $P \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$. The transformed systems $\text{LT}(\mathcal{R})$ and $\text{LT}^=(\mathcal{R}_\sim, P)$ over $\mathcal{F} \cup \{\text{top}, \text{ok}, \text{check}\}$ are defined as follows:

$$\boxed{\mathrm{LT}(\mathcal{R})}$$

$$\begin{array}{rcll} l & \to & r & \text{for all non-top rules } l \to r \text{ in } \mathcal{R} \\ \mathrm{top}(\mathrm{ok}(l)) & \to & \mathrm{top}(\mathrm{check}(r)) & \text{for all top rules } \mathrm{top}(l) \to \mathrm{top}(r) \text{ in } \mathcal{R} \end{array}$$

$$\boxed{\mathrm{LT}^{=}(\mathcal{R}_\sim, P)}$$

$$\begin{array}{rcll} l & \to & r & \text{for all non-top rules } l \to r \text{ in } \mathcal{R}_\sim \\ \mathrm{top}(\mathrm{ok}(l)) & \to & \mathrm{top}(\mathrm{check}(r)) & \text{for all top rules } \mathrm{top}(l) \to \mathrm{top}(r) \text{ in } \mathcal{R}_\sim \\ \mathrm{check}(p) & \to & \mathrm{ok}(p) & \text{for all } p \in P \\ f(x_1, \ldots, \mathrm{ok}(x_i), \ldots, x_n) & \to & \mathrm{ok}(f(x_1, \ldots, x_n)) & \text{for all } f \in \mathcal{F},\ 1 \leqslant i \leqslant \mathrm{arity}(f) \\ \mathrm{check}(f(x_1, \ldots, x_n)) & \to & f(x_1, \ldots, \mathrm{check}(x_i), \ldots, x_n) & \\ & & & \text{for all } f \in \mathcal{F},\ 1 \leqslant i \leqslant \mathrm{arity}(f) \end{array}$$ ◇

The idea behind this transformation is that the presence of an ok symbol at the root of the term is intended to indicate existence of an instance of $p \in P$. Every time a top rule is applied this ok symbol is transformed to a check symbol. This check symbol can traverse toward the leaves and upon reaching an instance of some term $p \in P$ is transformed back into an ok symbol. This ok symbol can move up again and allow further top reductions upon reaching the root of the term.

Few remarks concerning the transformation:

- For readability concerns we will write $\to_{\mathrm{LT}}$ instead of $\to_{\mathrm{LT}(\mathcal{R})}$ and $\tilde{\to}_{\mathrm{LT}}$ instead of $\to_{\mathrm{LT}^{=}(\mathcal{R}_\sim, P)}$.
- In order to apply automatic techniques the set P should be finite, otherwise the TRS $\mathrm{LT}^{=}(\mathcal{R}_\sim, P)$ is infinite.
- If the liveness problem does not involve fairness, so it is modeled by a single TRS $\mathcal{R}$, then we define the result of the transformation to be also a single TRS, namely $\mathrm{LT}^{=}(\mathcal{R}, P)$ and proving liveness comes down to showing termination, instead of relative termination, of the transformed system..

5.2.3 Soundness

Now we show soundness, that is relative termination of the transformed system implies liveness of the original one.

Theorem 5.9 (Soundness). *Let $\mathcal{R}, \mathcal{R}_\sim$ be top TRSs over $\mathcal{F} \cup \{\mathrm{top}\}$, let $P \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$. Then:*

$$\mathrm{SN}(\mathrm{LT}(\mathcal{R})/\mathrm{LT}^{=}(\mathcal{R}_\sim, P)) \implies \mathrm{Live}(\mathcal{T}_{\mathrm{top}}(\mathcal{F}, \mathcal{V}), \mathcal{R}, \mathcal{R}_\sim, G(P))$$

Proof. Assume that $\mathrm{SN}(\mathrm{LT}(\mathcal{R})/\mathrm{LT}^{=}(\mathcal{R}_{\sim}, P))$ holds and $\mathrm{Live}(\mathcal{T}_{\mathrm{top}}(\mathcal{F}, \mathcal{V}), \mathcal{R}, \mathcal{R}_{\sim}, G(P))$ does not hold. Then, by Theorem 5.4, $\mathrm{SN}(\mathcal{T}_{\mathrm{top}}(\mathcal{F}, \mathcal{V}), \rightarrow_{\mathcal{G}} / \tilde{\rightarrow}_{\mathcal{G}})$ does not hold as it is equivalent to the liveness problem in question. That means that there is an infinite $\rightarrow_{\mathcal{G}} / \tilde{\rightarrow}_{\mathcal{G}}$ reduction of top terms. We will show that this infinite reduction can be mapped to an infinite $\rightarrow_{\mathrm{LT}} / \tilde{\rightarrow}_{\mathrm{LT}}$ reduction, thus contradicting $\mathrm{SN}(\mathrm{LT}(\mathcal{R})/\mathrm{LT}^{=}(\mathcal{R}_{\sim}, P))$. For that purpose it is sufficient to show that:

$$\mathrm{top}(t) \rightarrow_{\mathcal{G}} / \tilde{\rightarrow}_{\mathcal{G}} \mathrm{top}(u) \implies \mathrm{top}(\mathrm{ok}(t)) \rightarrow_{\mathrm{LT}} / \tilde{\rightarrow}_{\mathrm{LT}} \mathrm{top}(\mathrm{ok}(u))$$

that is that any step in $\rightarrow_{\mathcal{G}} / \tilde{\rightarrow}_{\mathcal{G}}$ can be mimicked by a step in $\rightarrow_{\mathrm{LT}} / \tilde{\rightarrow}_{\mathrm{LT}}$. It easily follows if we can show that:

(i) whenever $\mathrm{top}(t) \rightarrow_{\mathcal{G}} \mathrm{top}(u)$ then $\mathrm{top}(\mathrm{ok}(t)) \rightarrow_{\mathrm{LT}} / \tilde{\rightarrow}_{\mathrm{LT}} \mathrm{top}(\mathrm{ok}(u))$, and

(ii) whenever $\mathrm{top}(t) \tilde{\rightarrow}_{\mathcal{G}} \mathrm{top}(u)$ then $\mathrm{top}(\mathrm{ok}(t)) \tilde{\rightarrow}^{*}_{\mathrm{LT}} \mathrm{top}(\mathrm{ok}(u))$.

To show (i) first observe that if $\mathrm{top}(t) \rightarrow_{\mathcal{G}} \mathrm{top}(u)$ by the application of a non-top rule $l \rightarrow r$ then the same rule is present in $\mathrm{LT}(\mathcal{R})$ so we trivially have $\mathrm{top}(\mathrm{ok}(t)) \rightarrow_{\mathrm{LT}} / \tilde{\rightarrow}_{\mathrm{LT}} \mathrm{top}(\mathrm{ok}(u))$.

If on the other hand $\mathrm{top}(t) \rightarrow_{\mathcal{G}} \mathrm{top}(u)$ by application of a top rule then from the definition of top TRSs we have that $t = l\sigma$ and $u = r\sigma$ for some substitution σ and some rule $\mathrm{top}(l) \rightarrow \mathrm{top}(r)$ from $\mathcal{R}$. Note that $\mathrm{top}(u)$ is part of an infinite $\rightarrow_{G} / \tilde{\rightarrow}_{G}$ reduction so $\mathrm{top}(u) \rightarrow_{\mathcal{G}} \mathrm{top}(w)$ or $\mathrm{top}(u) \tilde{\rightarrow}_{\mathcal{G}} \mathrm{top}(w)$ for some term w. Then from the definition of $\rightarrow_{\mathcal{G}}$ we get that $\mathrm{top}(u)$ does contain an instance of some $p \in P$ which means that we have $u = C[p\gamma]$ for some context C and some substitution γ. Then we have:

$$\begin{array}{rcll}
\mathrm{top}(\mathrm{ok}(t)) & = & \mathrm{top}(\mathrm{ok}(l\sigma)) & \\
& \rightarrow_{\mathrm{LT}} & \mathrm{top}(\mathrm{check}(r\sigma)) & \\
& = & \mathrm{top}(\mathrm{check}(C[p\gamma])) & \\
& \tilde{\rightarrow}^{*}_{\mathrm{LT}} & \mathrm{top}(C[\mathrm{check}(p)\gamma]) & \\
& \tilde{\rightarrow}_{\mathrm{LT}} & \mathrm{top}(C[\mathrm{ok}(p)\gamma]) & \\
& \tilde{\rightarrow}^{*}_{\mathrm{LT}} & \mathrm{top}(\mathrm{ok}(C[p\gamma])) & = \mathrm{top}(\mathrm{ok}(u))
\end{array}$$

The reasoning for (ii) is similar, except that the first step is from $\tilde{\rightarrow}_{\mathrm{LT}(\mathcal{R})}$ instead of $\rightarrow_{\mathrm{LT}(\mathcal{R})}$. □

5.2.4 Completeness

In the previous subsection we proved that our approach is correct, that is that the proposed transformation is sound. Now we will try to address the question of its power. First in the following theorem we show that any liveness problem that could be dealt with using LS can also be dealt with using LT. Then we show that under some restrictions our approach is even complete.

Theorem 5.10. *Let* $\mathcal{R}$ *be a top TRS over* $\mathcal{F} \cup \{\text{top}\}$ *and* $p \in \mathcal{T}(\mathcal{F}, \mathcal{V})$*. Then* $\text{SN}(\text{LS}(\mathcal{R}, p))$ *implies* $\text{SN}(\text{LT}^{=}(\mathcal{R}, \{p\}))$*.*

Proof. Assume $\text{SN}(\text{LS}(\mathcal{R}, p))$ and not $\text{SN}(\text{LT}^{=}(\mathcal{R}, \{p\}))$. Then there is an infinite reduction sequence in $\text{LT}^{=}(\mathcal{R}, \{p\})$. It follows easily from the definitions of $\text{LT}^{=}$ and LS that by dropping all ok symbols from terms in this reduction we obtain a valid reduction sequence in $\text{LS}(\mathcal{R}, p)$. Only applications of

$$f(x_1, \ldots, \text{ok}(x_i), \ldots, x_n) \to \text{ok}(f(x_1, \ldots, x_n))$$

become equalities but since this rule itself is terminating, in that infinite reduction there must be infinitely many steps not using this rule. So in this way we map an infinite reduction in $\text{LT}^{=}(\mathcal{R}, \{p\})$ to an infinite reduction in $\text{LS}(\mathcal{R}, p)$, thus contradicting $\text{SN}(\text{LS}(\mathcal{R}, p))$. $\square$

Note however that there is no ‘if and only if’ in Theorem 5.10, which means that LT may succeed in case LS fails, as illustrated in the following example:

Example 5.11. *Consider the TRS*

$$\begin{aligned} \text{top}(\text{f}(x, \text{b})) &\to \text{top}(\text{f}(\text{b}, \text{b})) \\ \text{a} &\to \text{b} \end{aligned}$$

from Example 5.7, where we concluded that $\text{LS}(\mathcal{R}, p)$ *applied on this example yields a not terminating TRS. However it is not difficult to see that* $\text{LT}^{=}(\mathcal{R}, \{p\})$ *is terminating. Two more complex and practical examples will be presented in Section 5.3.* $\lhd$

There is a good reason why the sound and complete transformation presented in [GZ03a] is so complicated, so clearly enough we cannot hope that a transformation as simple as LT would be complete. The best we can hope for is completeness under some additional restrictions on the shape of TRSs modeling the liveness problem. Indeed that is the case. First we present three such requirements along with examples showing that if they do not hold completeness is lost. However, for the setting of liveness problems, these requirements are quite mild. Then we will prove completeness for the restricted set of systems for which they do hold.

Example 5.12. *Let us begin with a very simple example, namely:*

$$\begin{aligned} \text{top}(\text{a}) &\to \text{top}(\text{b}) \\ \text{b} &\to \text{a} \end{aligned}$$

Consider liveness with $P = \{\text{a}\}$*, meaning that in every reduction eventually a term without* a *is reached. It is an easy observation that in every infinite reduction of* *this TRS its two rules have to be applied interchangeably, so after at most one step the term without a is reached and liveness holds. But the transformation yields:*

$$\mathrm{LT}(\mathcal{R}) = \{\mathrm{top}(\mathrm{ok}(\mathrm{a})) \to \mathrm{top}(\mathrm{check}(\mathrm{b})),\ \mathrm{b} \to \mathrm{a}\}$$
$$\mathrm{LT}^{=}(\mathcal{R}_\sim, P) = \{\mathrm{check}(\mathrm{a}) \xrightarrow{\sim} \mathrm{ok}(\mathrm{a})\}$$

The above system allows an infinite $\to_{\mathrm{LT}} / \xrightarrow{\sim}_{\mathrm{LT}}$ reduction, namely:

$$\mathrm{top}(\mathrm{ok}(\mathrm{a})) \to_{\mathrm{LT}} \mathrm{top}(\mathrm{check}(\mathrm{b})) \to_{\mathrm{LT}} \mathrm{top}(\mathrm{check}(\mathrm{a})) \xrightarrow{\sim}_{\mathrm{LT}} \mathrm{top}(\mathrm{ok}(\mathrm{a})) \to_{\mathrm{LT}} \dots$$

The reason why things go wrong here is that some term from P (being a *in this case) can be created, that is there are reductions from terms not containing an instance of p (for any $p \in P$) to terms containing an instance of p (for some $p \in P$). We can mend that by forbidding this kind of behavior. Let us note that this means restricting to liveness problems for which if the desired property holds at some point it will hold from that point onwards.* ◁

From now on, for readability concerns, we will assume that rules from $\mathcal{R}$ are given as $l \to r$ and rules from $\mathcal{R}_\sim$ as $l \xrightarrow{\sim} r$ and, following the notation introduced in Section 1.2, we will just present a set of rules leaving the separation to $\mathcal{R}$ and $\mathcal{R}_\sim$ implicit. Now we will show another property that can destroy liveness.

Example 5.13. *Consider the following TRS over* $\{\mathrm{f}, \mathrm{g}, \mathrm{top}, \mathrm{a}, \mathrm{b}\}$*:*

$$\mathrm{top}(\mathrm{g}(x, y, \mathrm{a})) \to \mathrm{top}(\mathrm{f}(x))$$
$$\mathrm{f}(x) \to \mathrm{g}(x, x, x)$$

In any infinite top reduction the second rule is applied infinitely often, and a straightforward analysis shows that after applying the second rule in a top reduction, no infinite reduction from a term containing the symbol b *is possible. So liveness with $P = \{\mathrm{b}\}$ holds. The transformed system reads:*

$$
\begin{array}{rrcl}
(1) & \mathrm{top}(\mathrm{ok}(\mathrm{g}(x, y, \mathrm{a}))) & \to & \mathrm{top}(\mathrm{check}(\mathrm{f}(x)))\\
(2) & \mathrm{f}(x) & \to & \mathrm{g}(x, x, x)\\
(3) & \mathrm{check}(\mathrm{b}) & \xrightarrow{\sim} & \mathrm{ok}(\mathrm{b})\\
(4) & \mathrm{check}(\mathrm{f}(x)) & \xrightarrow{\sim} & \mathrm{f}(\mathrm{check}(x))\\
(5) & \mathrm{check}(\mathrm{g}(x, y, z)) & \xrightarrow{\sim} & \mathrm{g}(\mathrm{check}(x), y, z)\\
(6) & \mathrm{check}(\mathrm{g}(x, y, z)) & \xrightarrow{\sim} & \mathrm{g}(x, \mathrm{check}(y), z)
\end{array}
$$

$$
\begin{array}{rrcl}
(7) & \mathrm{check}(\mathrm{g}(x, y, z)) & \xrightarrow{\sim} & \mathrm{g}(x, y, \mathrm{check}(z))\\
(8) & \mathrm{f}(\mathrm{ok}(x)) & \xrightarrow{\sim} & \mathrm{ok}(\mathrm{f}(x))\\
(9) & \mathrm{g}(\mathrm{ok}(x), y, z) & \xrightarrow{\sim} & \mathrm{ok}(\mathrm{g}(x, y, z))\\
(10) & \mathrm{g}(x, \mathrm{ok}(y), z) & \xrightarrow{\sim} & \mathrm{ok}(\mathrm{g}(x, y, z))\\
(11) & \mathrm{g}(x, y, \mathrm{ok}(z)) & \xrightarrow{\sim} & \mathrm{ok}(\mathrm{g}(x, y, z))
\end{array}
$$

and allows an infinite reduction, namely:

$$\begin{aligned}
&\mathrm{top}(\mathrm{check}(\mathrm{f}(\mathrm{ok}(\mathrm{a})))) \xrightarrow{(2)} \mathrm{top}(\mathrm{check}(\mathrm{g}(\mathrm{ok}(\mathrm{a}), \mathrm{ok}(\mathrm{a}), \mathrm{ok}(\mathrm{a})))) \overset{(6)}{\tilde{\to}} \\
\mathrm{top}(\mathrm{g}(\mathrm{ok}(\mathrm{a}), \mathrm{check}(\mathrm{ok}(\mathrm{a})), \mathrm{ok}(\mathrm{a}))) &\overset{(11)}{\tilde{\to}} \mathrm{top}(\mathrm{ok}(\mathrm{g}(\mathrm{ok}(\mathrm{a}), \mathrm{check}(\mathrm{ok}(\mathrm{a})), \mathrm{a}))) \xrightarrow{(1)} \\
&\mathrm{top}(\mathrm{check}(\mathrm{f}(\mathrm{ok}(\mathrm{a})))) \to \dots
\end{aligned}$$

In this example completeness was harmed by the presence of duplicating rules in the original system. ◁

Example 5.14. *Finally consider the simple TRS over* $\{\mathrm{a}, \mathrm{b}, \mathrm{f}\}$ *consisting of the following two rules:*

$$\begin{aligned}
\mathrm{f}(\mathrm{a}) &\to \mathrm{b} \\
\mathrm{b} &\to \mathrm{b}
\end{aligned}$$

Clearly liveness with $P = \{\mathrm{a}\}$ *holds but after transformation we obtain:*

$$\begin{aligned}
\mathrm{f}(\mathrm{a}) &\to \mathrm{b} \\
\mathrm{b} &\to \mathrm{b} \\
\mathrm{check}(\mathrm{a}) &\tilde{\to} \mathrm{ok}(\mathrm{a})
\end{aligned}$$

being non-terminating since b *rewrites to itself.*

This gives rise to the third, and last, requirement, namely that the signature of the TRS for which we consider liveness problem must contain at least one symbol of arity $\geqslant 2$*. The precise reason for this requirement will become clear in the proof of Theorem 5.21. For now, note only that if a symbol* g *of arity* 2 *was present in our example then liveness would be lost as the term* $\mathrm{g}(\mathrm{a}, \mathrm{b})$ *rewrites to itself forming an infinite reduction containing* a*. Note that this is a weak requirement: we only require presence of a symbol of arity* $\geqslant 2$ *in the signature; it is not required that this symbol occurs in the rewrite rules.* ◁

Now we will prove that if all three above restrictions are satisfied, that is there are no duplicating rules, terms from P cannot be created and $\mathcal{F}$ contains some symbol of arity $\geqslant 2$, then the completeness holds.

Before we state the completeness theorem we need some auxiliary results. First let us denote by $\overline{t}$ the term t after removing all occurrences of ok and check symbols.

Definition 5.15. Let t be an arbitrary term over $\mathcal{F} \cup \{\mathrm{check}, \mathrm{ok}\}$. We define the mapping to a term $\overline{t}$ over $\mathcal{F}$ as follows:

$$\begin{aligned}
\overline{x} &= x \\
\overline{\mathrm{check}(t)} &= \overline{t} \\
\overline{\mathrm{ok}(t)} &= \overline{t} \\
\overline{f(t_1, \dots, t_n)} &= f(\overline{t_1}, \dots, \overline{t_n}) \quad \text{for } f \notin \{\mathrm{check}, \mathrm{ok}\}
\end{aligned}$$ ◇

We need two lemmas. First we will show that the reduction steps in a transformed system can be mimicked in the original system after removing extra ok and check symbols.

Lemma 5.16. *Given two TRSs $\mathcal{R}$ and $\mathcal{R}_\sim$ over the same signature $\mathcal{F}$ and arbitrary terms t, u, we have the following implications:*

(i) $t \to_{\mathrm{LT}} u \implies \overline{t} \to \overline{u}$

(ii) $t \tilde{\to}_{\mathrm{LT}} u \implies \overline{t} \tilde{\to}^* \overline{u}$

Proof. (i) It follows from an easy observation that for any rule $l \to r \in \mathrm{LT}(\mathcal{R})$ we have $\overline{l} \to \overline{r} \in \mathcal{R}$.

(ii) Similarly if $l \to r \in \mathrm{LT}^=(\mathcal{R}_\sim, P)$ then either $\overline{l} \to \overline{r} \in \mathcal{R}_\sim$ or $\overline{l} = \overline{r}$, but in any way $\overline{t} \tilde{\to}^* \overline{u}$. ☐

Later on we will need the following lemma stating that extending TRS with administrative rules for check and ok preserves termination.

Lemma 5.17. *Given two TRSs $\mathcal{R}$ and $\mathcal{R}_\sim$ over $\mathcal{F}$ $(\mathrm{top}, \mathrm{ok}, \mathrm{check} \notin \mathcal{F})$. Let $\mathcal{S}$ consist of the following rules:*

$$\begin{array}{rcll}
\mathrm{check}(p) & \to & \mathrm{ok}(p) & \textit{for all } p \in P \\
\mathrm{check}(f(x_1, \ldots, x_n)) & \to & f(x_1, \ldots, \mathrm{check}(x_i), \ldots, x_n) & \textit{for all } f \in \mathcal{F}, \\
& & & 1 \leqslant i \leqslant \mathrm{arity}(f) \\
f(x_1, \ldots, \mathrm{ok}(x_i), \ldots, x_n)) & \to & \mathrm{ok}(f(x_1, \ldots, x_n)) & \textit{for all } f \in \mathcal{F}, \\
& & & 1 \leqslant i \leqslant \mathrm{arity}(f)
\end{array}$$

Now if $\mathrm{SN}(\mathcal{R}/\mathcal{R}_\sim)$ *then* $\mathrm{SN}(\mathcal{R}/(\mathcal{R}_\sim \cup S))$.

Proof. Assume $\mathrm{SN}(\mathcal{R}/\mathcal{R}_\sim)$ and not $\mathrm{SN}(\mathcal{R}/\mathcal{R}_\sim \cup \mathcal{S})$. So there is an infinite $\to_\mathcal{R}$ / $\to_{\mathcal{R}_\sim} \cup \to_\mathcal{S}$ reduction. Because both $\mathcal{R}$ and $\mathcal{R}_\sim$ do not contain top symbol then $\mathcal{R} = \mathrm{LT}(\mathcal{R})$ and $\mathcal{R}_\sim \cup \mathcal{S} = \mathrm{LT}^=(\mathcal{R}_\sim, P)$. So with the use of Lemma 5.16 we can map this infinite $\to_{\mathrm{LT}(\mathcal{R})}$ / $\to_{\mathrm{LT}^=(\mathcal{R}_\sim, P)}$ reduction to infinite $\to_\mathcal{R}$ / $\to_{\mathcal{R}_\sim}$ reduction which contradicts $\mathrm{SN}(\mathcal{R}/\mathcal{R}_\sim)$. ☐

Now we will present the theorem stating that for non-duplicating TRSs relative termination on top terms is equivalent to relative termination on arbitrary terms. We start by an example showing that non-duplication is essential for that.

Example 5.18. *Let us consider the following TRS:*

$$\begin{array}{r}
\mathrm{top}(\mathrm{f}(x)) \to \mathrm{top}(\mathrm{a}) \\
\mathrm{f}(x) \tilde{\to} \mathrm{g}(\mathrm{f}(x), \mathrm{f}(x))
\end{array}$$

Here relative termination on top terms follows from the observation that any $\rightarrow$*-step on any top term always yields the normal form* top(a)*. However, this system admits the following fair reduction:*

$$\mathrm{f}(\mathrm{top}(\mathrm{f}(x))) \overset{\sim}{\rightarrow} \mathrm{g}(\mathrm{f}(\mathrm{top}(\mathrm{f}(x))), \mathrm{f}(\mathrm{top}(\mathrm{f}(x)))) \rightarrow \mathrm{g}(\mathrm{f}(\mathrm{top}(\mathrm{a}), \underbrace{\mathrm{f}(\mathrm{top}(\mathrm{f}(x)))}_{\textit{initial term}}) \overset{\sim}{\rightarrow} \cdots . \quad \lhd$$

Next we prove that in case of non-duplication the above mentioned equivalence does hold. First we need a simple lemma.

Lemma 5.19. *Let* $\mathcal{R}$ *be a non-duplicating top TRS over* $\mathcal{F}$*. Let* C, D *be* n*-hole contexts not containing the symbol* top*, and let* $t_1, \ldots, t_n, u_1, \ldots, u_n$ *be terms possibly containing* top *for which*

$$\mathrm{top}(C[\mathrm{top}(t_1), \ldots, \mathrm{top}(t_n)]) \rightarrow_{\mathcal{R}} \mathrm{top}(D[\mathrm{top}(u_1), \ldots, \mathrm{top}(u_n)])$$

Then either

- $C = D$ *and* $\mathrm{top}(t_i) \rightarrow_{\mathcal{R}} \mathrm{top}(u_i)$ *for some* i *and* $t_j = u_j$ *for all* $j \neq i$*, or*
- $\mathrm{top}(C[x, \ldots, x]) \rightarrow_{\mathcal{R}} \mathrm{top}(D[x, \ldots, x])$ *and the multiset* $\{\mathrm{top}(t_1), \ldots, \mathrm{top}(t_n)\}$ *is equal to the multiset* $\{\mathrm{top}(u_1), \ldots, \mathrm{top}(u_n)\}$*.*

Proof. The top symbol can occur only as a root of the $\mathcal{R}$ rules, hence $\rightarrow_{\mathcal{R}}$ reduction step can occur only in one of the n holes of the context, or in the context itself (it is not possible for the redex to overlap between the context and its holes). If it is in the i'th hole, then the first bullet is satisfied, as indeed we have $\mathrm{top}(t_i) \rightarrow_{\mathcal{R}} \mathrm{top}(u_i)$ and the remaining holes and the context remain unchanged. If, on the other hand, the reduction is in the context then indeed $\mathrm{top}(C[x, \ldots, x]) \rightarrow_{\mathcal{R}} \mathrm{top}(D[x, \ldots, x])$ and the multisets $\{\mathrm{top}(t_1), \ldots, \mathrm{top}(t_n)\}$ and $\{\mathrm{top}(u_1), \ldots, \mathrm{top}(u_n)\}$ are equal, as all the terms occurring in those multisets must be in the substitution part of the rule application, both contexts have n holes and $\mathcal{R}$ is non-duplicating. $\square$

Now we formulate the theorem stating equivalence of relative termination on top terms with relative termination on arbitrary terms for non-duplicating TRSs.

Theorem 5.20. *Let* $\mathcal{R}, \mathcal{R}_\sim$ *be non-duplicating top TRSs over* $\mathcal{F}$ *Then we have:*

$$\mathrm{SN}(\mathcal{T}(\mathcal{F}, \mathcal{V}), \mathcal{R}/\mathcal{R}_\sim) \iff \mathrm{SN}(\mathcal{T}_{\mathrm{top}}(\mathcal{F}, \mathcal{V}), \mathcal{R}/\mathcal{R}_\sim)$$

Proof. The ($\Rightarrow$)-part is trivial. For the ($\Leftarrow$)-part assume we have an arbitrary infinite fair reduction; we have to prove that there is also an infinite fair top reduction. By putting a top symbol around all terms we force that all terms in the infinite fair reduction have top as the root symbol. Next among all infinite fair reductions having top as the root symbol we choose one in which the number

N of top symbols occurring in the initial term is minimal. Due to non-duplication in every term in this reduction at most N top symbols occur; due to minimality of N we conclude that each of these terms contains exactly N top symbols. Write $\mathrm{top}(C[\mathrm{top}(t_1), \ldots, \mathrm{top}(t_n)])$ for the initial term in the reduction for a context C not containing the symbol top. Since the number of top-symbols remains unchanged every term in the reduction is of the same shape, having the same number n of holes in the context. Due to minimality every infinite $\rightarrow \cup \stackrel{\sim}{\rightarrow}$ reduction of $\mathrm{top}(t_i)$ contains only finitely many $\rightarrow$-steps, for $i = 1, \ldots, n$. Due to Lemma 5.19 all steps are either in the context or in descendants of $\mathrm{top}(t_i)$. Since the descendants of $\mathrm{top}(t_i)$ allow only finitely many $\rightarrow$-steps and there are infinitely many $\rightarrow$-steps in total, we conclude that there are infinitely many $\rightarrow$-steps in the contexts. More precisely, we arrive at an infinite top reduction of $\mathrm{top}(C[x, \ldots, x])$ containing infinitely many $\rightarrow$-steps, contradicting $\mathrm{SN}(\mathcal{T}_{\mathrm{top}}(\mathcal{F}, \mathcal{V}), \mathcal{R}/\mathcal{R}_{\sim})$. $\square$

Now we formulate the theorem which states that, under the three extra requirements introduced before, the transformation defined in Section 5.2.2 is complete.

Theorem 5.21 (Completeness)**.** *Let $\mathcal{R}$, $\mathcal{R}_{\sim}$ be top TRSs over $\mathcal{F} \cup \{\mathrm{top}\}$. If the following conditions are satisfied:*

(i) if u contains an instance of some $p \in P$ and $t \rightarrow u$ or $t \stackrel{\sim}{\rightarrow} u$ then t also contains an instance of p,

(ii) both $\mathcal{R}$ and $\mathcal{R}_{\sim}$ are non-duplicating,

(iii) there is at least one function symbol of arity $\geqslant 2$ in $\mathcal{F}$.

Then:

$$\mathrm{Live}(\mathcal{T}_{\mathrm{top}}(\mathcal{F}, \mathcal{V}), \mathcal{R}, \mathcal{R}_{\sim}, G(P)) \implies \mathrm{SN}(\mathrm{LT}(\mathcal{R})/\,\mathrm{LT}^{=}(\mathcal{R}_{\sim}, P))$$

Proof. Assume that $\mathrm{Live}(\mathcal{T}_{\mathrm{top}}(\mathcal{F}, \mathcal{V}), \mathcal{R}, \mathcal{R}_{\sim}, G(P))$ and conditions (i)-(iii) hold and $\mathrm{SN}(\mathrm{LT}(\mathcal{R})/\,\mathrm{LT}^{=}(\mathcal{R}_{\sim}, P))$ does not hold. Then there is an infinite $\rightarrow_{\mathrm{LT}} / \stackrel{\sim}{\rightarrow}_{\mathrm{LT}}$ reduction. Due to non-duplication of $\mathcal{R}$ and $\mathcal{R}_{\sim}$, $\mathrm{LT}(\mathcal{R})$ and $\mathrm{LT}^{=}(\mathcal{R}_{\sim}, P)$ are also non-duplicating and by application of Theorem 5.20 we get that there is an infinite top $\rightarrow_{\mathrm{LT}} / \stackrel{\sim}{\rightarrow}_{\mathrm{LT}}$ reduction, call it ω.

Assume infinitely many terms in ω contain instances of terms from P. By the observation that an instance of p occurs in $\overline{C[p\delta]}$, Lemma 5.16 applied to ω gives rise to an infinite top reduction in $\mathcal{R}/\mathcal{R}_{\sim}$ having infinitely many terms containing instances of $p \in P$. Due to (i) all terms in this infinite reduction contain instances of $p \in P$ contradicting $\mathrm{Live}(\mathcal{T}_{\mathrm{top}}(\mathcal{F}, \mathcal{V}), \mathcal{R}, \mathcal{R}_{\sim}, G(P))$. Hence only finitely many terms in ω contain instances of terms from P. So removing this finite prefix of ω yields an infinite top $\rightarrow_{\mathrm{LT}} / \stackrel{\sim}{\rightarrow}_{\mathrm{LT}}$ reduction ω' in which no instance of $p \in P$ occurs at all.

Note that non-top rules of $\mathcal{R}$ are relatively terminating to non-top rules of $\mathcal{R}_\sim$. Assume they are not. Then there is an infinite $\rightarrow / \tilde{\rightarrow}$ reduction sequence obtained using non-top rules of $\mathcal{R}$ and $\mathcal{R}_\sim$. Let $f \in \mathcal{F}$ be a function symbol of arity $\geqslant 2$ (its existence is ensured by (iii)). Put the infinite $\rightarrow / \tilde{\rightarrow}$ reduction in context $\mathrm{top}(f(p, \square, \ldots))$. This yields an infinite, fair top reduction containing p and thus contradicting $\mathrm{Live}(\mathcal{T}_{\mathrm{top}}(\mathcal{F}, \mathcal{V}), \mathcal{R}, \mathcal{R}_\sim, G(P))$. Now, by application of Lemma 5.17, we conclude that non-top rules of $\mathrm{LT}(\mathcal{R})$ are relatively terminating to non-top rules of $\mathrm{LT}^=(\mathcal{R}_\sim, P)$.

In ω' top rules are applied infinitely often as non-top rules of $\mathrm{LT}(\mathcal{R})$ are relatively terminating to non-top rules of $\mathrm{LT}^=(\mathcal{R}_\sim, P)$. Note that because of (ii) the only way to create an ok symbol is by application of the rule $\mathrm{check}(p) \tilde{\rightarrow} \mathrm{ok}(p)$. Every top reduction removes one occurrence of the ok symbol, so the aforementioned rule should be applied infinitely often. But since p does not occur in ω' this rule is not applicable which leads to a contradiction and ends the proof. $\square$

Examples 5.12, 5.13 and 5.14 show that conditions (i)-(iii) of this theorem are essential.

5.3 Examples

In this section we present two examples illustrating the applicability of the proposed transformation. None of them could be treated with the use of the LS transformation described in [GZ03a]. Both relative termination proofs of the transformed systems were found completely automatically by TPA.

Example 5.22. *[Cars over a bridge] There is a road with cars going in two directions. But on their way there is a bridge which is only wide enough to permit a single lane of traffic. So there are lights indicating which side of the bridge is allowed to cross it. We want to verify the liveness property, namely that every car will eventually be able to cross the bridge. For that clearly we need some assumptions about the lighting system. We want to be as general as possible so instead of assuming some particular algorithm of switching lights we just require them to change in a fair way, that is in the infinite observation of the system there must be infinitely many light switches (or equivalently: no matter when we start watching the road after some, arbitrary, time we will see the change of lights). Also we assume that before a light switches, if there are cars waiting at least one of them will pass (otherwise liveness is lost as lights can change all the time without any cars passing).*

We model the system with a unary top *symbol whose arguments start with a binary symbol* left *or* right *indicating which side has a green light. The arguments of* left *and* right *start with unary symbols* new *and* old *representing cars waiting to cross the bridge. The constant* bot *stands for the end of the queue. New cars are allowed* *to arrive at the end of the queue at any time. What we want to prove is that finally no old car remains.*

$$
\begin{array}{rrcl}
(1) & \mathrm{top}(\mathrm{left}(\mathrm{old}(x)), y) & \to & \mathrm{top}(\mathrm{right}(x, y)) \\
(2) & \mathrm{top}(\mathrm{left}(\mathrm{new}(x)), y) & \to & \mathrm{top}(\mathrm{right}(x, y)) \\
(3) & \mathrm{top}(\mathrm{right}(x, \mathrm{old}(y))) & \to & \mathrm{top}(\mathrm{left}(x, y)) \\
(4) & \mathrm{top}(\mathrm{right}(x, \mathrm{new}(y))) & \to & \mathrm{top}(\mathrm{left}(x, y)) \\
(5) & \mathrm{top}(\mathrm{left}(\mathrm{bot}, y)) & \to & \mathrm{top}(\mathrm{right}(\mathrm{bot}, y)) \\
(6) & \mathrm{top}(\mathrm{right}(x, \mathrm{bot})) & \to & \mathrm{top}(\mathrm{left}(x, \mathrm{bot})) \\
(7) & \mathrm{top}(\mathrm{left}(\mathrm{old}(x), y)) & \overset{\sim}{\to} & \mathrm{top}(\mathrm{left}(x, y)) \\
(8) & \mathrm{top}(\mathrm{left}(\mathrm{new}(x), y)) & \overset{\sim}{\to} & \mathrm{top}(\mathrm{left}(x, y)) \\
(9) & \mathrm{top}(\mathrm{right}(x, \mathrm{old}(y))) & \overset{\sim}{\to} & \mathrm{top}(\mathrm{right}(x, y)) \\
(10) & \mathrm{top}(\mathrm{right}(x, \mathrm{new}(y))) & \overset{\sim}{\to} & \mathrm{top}(\mathrm{right}(x, y)) \\
(11) & \mathrm{bot} & \overset{\sim}{\to} & \mathrm{new}(\mathrm{bot})
\end{array}
$$

We have the following semantics of the rules:

Rules	*Semantics*
(1) − (4)	*Car passes and the light changes.*
(5) − (6)	*No car waiting, light can change.*
(7) − (10)	*Car passes, light remains the same.*
(11)	*New car arriving.*

Using the transformation as described in Sect. 5.2.2 with $P = \{\mathrm{old}(x)\}$ *we obtain:*

$$
\begin{array}{rcl}
\mathrm{top}(\mathrm{ok}(\mathrm{left}(\mathrm{old}(x), y))) & \to & \mathrm{top}(\mathrm{check}(\mathrm{right}(x, y))) \\
\mathrm{top}(\mathrm{ok}(\mathrm{left}(\mathrm{new}(x), y))) & \to & \mathrm{top}(\mathrm{check}(\mathrm{right}(x, y))) \\
\mathrm{top}(\mathrm{ok}(\mathrm{right}(x, \mathrm{old}(y)))) & \to & \mathrm{top}(\mathrm{check}(\mathrm{left}(x, y))) \\
\mathrm{top}(\mathrm{right}(x, \mathrm{new}(y))) & \to & \mathrm{top}(\mathrm{check}(\mathrm{left}(x, y))) \\
\mathrm{top}(\mathrm{ok}(\mathrm{left}(\mathrm{bot}, y))) & \to & \mathrm{top}(\mathrm{check}(\mathrm{right}(\mathrm{bot}, y))) \\
\mathrm{top}(\mathrm{ok}(\mathrm{right}(x, \mathrm{bot}))) & \to & \mathrm{top}(\mathrm{check}(\mathrm{left}(x, \mathrm{bot}))) \\
\mathrm{top}(\mathrm{ok}(\mathrm{left}(\mathrm{old}(x), y))) & \overset{\sim}{\to} & \mathrm{top}(\mathrm{check}(\mathrm{left}(x, y))) \\
\mathrm{top}(\mathrm{ok}(\mathrm{left}(\mathrm{new}(x), y))) & \overset{\sim}{\to} & \mathrm{top}(\mathrm{check}(\mathrm{left}(x, y))) \\
\mathrm{top}(\mathrm{ok}(\mathrm{right}(x, \mathrm{old}(y)))) & \overset{\sim}{\to} & \mathrm{top}(\mathrm{check}(\mathrm{right}(x, y))) \\
\mathrm{top}(\mathrm{ok}(\mathrm{right}(x, \mathrm{new}(y)))) & \overset{\sim}{\to} & \mathrm{top}(\mathrm{check}(\mathrm{right}(x, y)))
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{bot} & \overset{\sim}{\to} & \mathrm{new}(\mathrm{bot}) \\
\mathrm{new}(\mathrm{ok}(x)) & \overset{\sim}{\to} & \mathrm{ok}(\mathrm{new}(x)) \\
\mathrm{old}(\mathrm{ok}(x)) & \overset{\sim}{\to} & \mathrm{ok}(\mathrm{old}(x)) \\
\mathrm{left}(\mathrm{ok}(x), y) & \overset{\sim}{\to} & \mathrm{ok}(\mathrm{left}(x, y)) \\
\mathrm{left}(x, \mathrm{ok}(y)) & \overset{\sim}{\to} & \mathrm{ok}(\mathrm{left}(x, y)) \\
\mathrm{right}(\mathrm{ok}(x), y) & \overset{\sim}{\to} & \mathrm{ok}(\mathrm{right}(x, y)) \\
\mathrm{right}(x, \mathrm{ok}(y)) & \overset{\sim}{\to} & \mathrm{ok}(\mathrm{right}(x, y)) \\
\mathrm{check}(\mathrm{old}(x)) & \overset{\sim}{\to} & \mathrm{ok}(\mathrm{old}(x)) \\
\mathrm{check}(\mathrm{new}(x)) & \overset{\sim}{\to} & \mathrm{new}(\mathrm{check}(x)) \\
\mathrm{check}(\mathrm{left}(x, y)) & \overset{\sim}{\to} & \mathrm{left}(\mathrm{check}(x), y) \\
\mathrm{check}(\mathrm{left}(x, y)) & \overset{\sim}{\to} & \mathrm{left}(x, \mathrm{check}(y)) \\
\mathrm{check}(\mathrm{right}(x, y)) & \overset{\sim}{\to} & \mathrm{right}(\mathrm{check}(x), y) \\
\mathrm{check}(\mathrm{right}(x, y)) & \overset{\sim}{\to} & \mathrm{right}(x, \mathrm{check}(y))
\end{array}
$$

All requirements of Theorem 5.21 are satisfied which means that by answering, either positively or negatively, the question of relative termination of the above TRS we can, respectively, prove or disapprove the liveness property.

It is an easy observation that the following procedure is termination-preserving: if for every rule the number of occurrences of some symbol is bigger or equal in the left hand side than in the right hand side, then remove the rules for which it is strictly bigger. This approach, already presented in [GZ03a], corresponds to proving termination with polynomial orderings with successor as interpretation for symbol begin counted and identity for all the other symbols.

The proof of relative termination can be given as follows. First count occurrences of old *to remove four rules. Then apply semantic labeling over* $\{0, 1\}$ *taking constant* 1 *for* old*, identity for remaining unary symbols, disjunction for all binary symbols and constant* 0 *for* bot*. In the resulting system repeatedly apply counting argument to remove all the* $\rightarrow$ *rules thus proving relative termination. The detailed, automatically generated relative termination proof can be found in [KZ05].* $\lhd$

The next example we investigate is commonly known as "the readers-writers problem" and goes back to Courtois et al. [CHP71]. It is considered as a classical synchronization problem.

Example 5.23. *[The readers and writers problem] Some resource is to be shared among a number of processes. There are two types of processes: "readers", which perform only reading operation and "writers" which can perform both reading and writing. The safety requirement is that writers must have exclusive access to the resource (that is when a writer has access to the resource no other process can have it) whereas readers can share the access (as long as there is no writer active at the same time).*

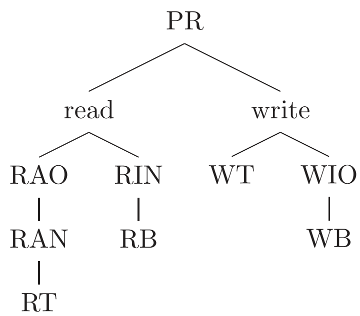
It is usual in literature ([SGG04], [RB86]) to concentrate only on safety requirements and propose a solution with priority for readers (writers) which can clearly lead to starvation of writers (readers). In [Hoa74] a fair solution to this problem has been proposed. We will present another variant of this starvation-free solution, where the access to the resource is controlled in a first-come first-served manner and we will verify that indeed starvation is not possible, corresponding to liveness.

To achieve that we introduce a flag indicating which group of processes (either readers or writers) has priority. If only one group claims the resource it is simply allowed to use it. But in case of a conflict, that is two groups interested in the use of the resource, the group having priority is allowed to access it and then the priority is changed. Without adding this priority flag obviously the desired liveness property does not hold.

As in Example 5.22 we distinguish between old and new processes and verify that finally there are no old processes in the system. We model reader processes by unary *function symbols:* RAO, RAN, RIO, RIN *where the second character indicates whether the reader is currently* ***A****ctive (performs reading) or* ***I****nactive (waits for access to the resource) and the third character indicates whether the reader is* ***O****ld or* ***N****ew. The argument is used to organize processes into lists. Similarly for writers we have* WAO, WAN, WIO, WIN*. However* WAO *and* WAN *are constants as there can be at most one active writing process at a time and there is no need to keep a list of such processes.*

The whole system is then modelled by means of binary function symbols PR *or* PW *indicating priority for readers or writers respectively. The first argument describes all readers in the system and the second one models writers. Readers are modelled by a binary operator* read *whose first argument contains the list of active processes terminated by constant* RT *and the second argument contains the list of processes waiting for the resource terminated by constant* RB*. Similarly, the binary operator* write *describes writers processes. Its first argument can be either* WT *(no active writer),* WAO *(active old writer) or* WAN *(active new writer). The second argument describes a list of inactive writers.*

Let us take a look at the following term tree:



It describes a system with two processes currently performing reading; one of them is old and the other is new. There is also one new reader waiting for the resource. There are no processes performing writing but there is one old writing process waiting for the resource. The priority is set for readers.

Due to using lists to represent active processes, we make one additional restriction that simplifies the modeling substantially, namely we assume that reading processes free the resource in the same order as they got access to it. It corresponds to the situation when the reading operation always takes some fixed interval of time. Now we are ready to present the model of the system.

$$
\begin{array}{llll}
(1) & \mathrm{RB} \overset{\sim}{\to} \mathrm{RIN}(\mathrm{RB}) & (4) & \mathrm{RAN}(\mathrm{RT}) \to \mathrm{RT} \\
(2) & \mathrm{WB} \overset{\sim}{\to} \mathrm{WIN}(\mathrm{WB}) & (5) & \mathrm{WAO} \to \mathrm{WT} \\
(3) & \mathrm{RAO}(\mathrm{RT}) \to \mathrm{RT} & (6) & \mathrm{WAN} \to \mathrm{WT}
\end{array}
$$

$$\mathrm{top}(\mathrm{PR}(\mathrm{read}(r_1, \mathrm{RIO}(r_2)), \mathrm{write}(\mathrm{WT}, \mathrm{WB}))) \overset{\sim}{\rightarrow} \mathrm{top}(\mathrm{PR}(\mathrm{read}(\mathrm{RAO}(r_1), r_2), \mathrm{write}(\mathrm{WT}, \mathrm{WB}))) \tag{7}$$

$$\mathrm{top}(\mathrm{PW}(\mathrm{read}(r_1, \mathrm{RIO}(r_2)), \mathrm{write}(\mathrm{WT}, \mathrm{WB}))) \overset{\sim}{\rightarrow} \mathrm{top}(\mathrm{PW}(\mathrm{read}(\mathrm{RAO}(r_1) r_2), \mathrm{write}(\mathrm{WT}, \mathrm{WB}))) \tag{8}$$

$$\mathrm{top}(\mathrm{PR}(\mathrm{read}(r_1, \mathrm{RIN}(r_2)), \mathrm{write}(\mathrm{WT}, \mathrm{WB}))) \overset{\sim}{\rightarrow} \mathrm{top}(\mathrm{PR}(\mathrm{read}(\mathrm{RAN}(r_1), r_2), \mathrm{write}(\mathrm{WT}, \mathrm{WB}))) \tag{9}$$

$$\mathrm{top}(\mathrm{PW}(\mathrm{read}(r_1, \mathrm{RIN}(r_2)), \mathrm{write}(\mathrm{WT}, \mathrm{WB}))) \overset{\sim}{\rightarrow} \mathrm{top}(\mathrm{PW}(\mathrm{read}(\mathrm{RAN}(r_1), r_2), \mathrm{write}(\mathrm{WT}, \mathrm{WB}))) \tag{10}$$

$$\mathrm{top}(\mathrm{PR}(\mathrm{read}(\mathrm{RT}, \mathrm{RB}), \mathrm{write}(\mathrm{WT}, \mathrm{WIN}(w)))) \overset{\sim}{\rightarrow} \mathrm{top}(\mathrm{PR}(\mathrm{read}(\mathrm{RT}, \mathrm{RB}), \mathrm{write}(\mathrm{WAN}, w))) \tag{11}$$

$$\mathrm{top}(\mathrm{PW}(\mathrm{read}(\mathrm{RT}, \mathrm{RB}), \mathrm{write}(\mathrm{WT}, \mathrm{WIN}(w)))) \overset{\sim}{\rightarrow} \mathrm{top}(\mathrm{PW}(\mathrm{read}(\mathrm{RT}, \mathrm{RB}), \mathrm{write}(\mathrm{WAN}, w))) \tag{12}$$

$$\mathrm{top}(\mathrm{PR}(\mathrm{read}(\mathrm{RT}, \mathrm{RB}), \mathrm{write}(\mathrm{WT}, \mathrm{WIO}(w)))) \overset{\sim}{\rightarrow} \mathrm{top}(\mathrm{PR}(\mathrm{read}(\mathrm{RT}, \mathrm{RB}), \mathrm{write}(\mathrm{WAO}, w))) \tag{13}$$

$$\mathrm{top}(\mathrm{PW}(\mathrm{read}(\mathrm{RT}, \mathrm{RB}), \mathrm{write}(\mathrm{WT}, \mathrm{WIO}(w)))) \overset{\sim}{\rightarrow} \mathrm{top}(\mathrm{PW}(\mathrm{read}(\mathrm{RT}, \mathrm{RB}), \mathrm{write}(\mathrm{WAO}, w))) \tag{14}$$

$$\mathrm{top}(\mathrm{PR}(\mathrm{read}(r_1, \mathrm{RIO}(r_2)), \mathrm{write}(\mathrm{WT}, w))) \overset{\sim}{\rightarrow} \mathrm{top}(\mathrm{PW}(\mathrm{read}(\mathrm{RAO}(r_1), r_2), \mathrm{write}(\mathrm{WT}, w))) \tag{15}$$

$$\mathrm{top}(\mathrm{PR}(\mathrm{read}(r_1, \mathrm{RIN}(r_2)), \mathrm{write}(\mathrm{WT}, w))) \overset{\sim}{\rightarrow} \mathrm{top}(\mathrm{PW}(\mathrm{read}(\mathrm{RAN}(r_1), r_2), \mathrm{write}(\mathrm{WT}, w))) \tag{16}$$

$$\mathrm{top}(\mathrm{PW}(\mathrm{read}(\mathrm{RT}, r_2), \mathrm{write}(\mathrm{WT}, \mathrm{WIO}(w)))) \overset{\sim}{\rightarrow} \mathrm{top}(\mathrm{PR}(\mathrm{read}(\mathrm{RT}, r_2), \mathrm{write}(\mathrm{WAO}, w))) \tag{17}$$

$$\mathrm{top}(\mathrm{PW}(\mathrm{read}(\mathrm{RT}, r_2), \mathrm{write}(\mathrm{WT}, \mathrm{WIN}(w)))) \overset{\sim}{\rightarrow} \mathrm{top}(\mathrm{PR}(\mathrm{read}(\mathrm{RT}, r_2), \mathrm{write}(\mathrm{WAN}, w))) \tag{18}$$

The meaning of the rules is as follows:

Rules	*Semantics*
$(1-2)$	*New inactive process appears in the system and is queued to wait for the resource.*
$(3-6)$	*Active process finishes reading/writing.*
$(7-10)$	*Nobody is writing nor waiting for write access — inactive reading process is allowed to start reading; priority does not change*
$(11-14)$	*Nobody is reading nor waiting for read access and nobody is writing — writer is allowed to start writing; priority does not change.*
$(15-16)$	*Nobody is writing and priority is for readers — reader is allowed to start reading; priority is switched.*
$(17-18)$	*Nobody is reading nor writing and priority is for writers — writer is allowed to start writing; priority is switched.*

What we want to prove is that finally no old process remains in the system. This corresponds to verifying liveness with the set $P = \{\mathrm{RAO}(x), \mathrm{RIO}(x), \mathrm{WAO}, \mathrm{WIO}(x)\}$. *Application of the transformation yields a pair of TRSs* $\mathcal{R}, \mathcal{R}_\sim$ *containing 42 rules in total. The corresponding relative termination problem can be solved by* TPA *thus proving the liveness property for the readers-writers problem. For a more detailed description of the transformed system and the related relative termination proof we refer to [KZ05].* $\triangleleft$

5.4 Conclusions

This chapter describes a technique to transform some liveness problems with fairness to the problems of proving relative termination of a transformed TRS. In a number of examples the latter could be done fully automatically. The only human activity in this approach is modeling the original problem in the language of term rewriting. Typically, finding the proof of relative transformation of the transformed TRS may be a hard job, and the computer generated proofs may be complicated, and of a shape that it is unlikely that they are found by a human.

Typically, the liveness problems that we consider involve infinite state spaces in two ways: they are not about a single set of initial states but involve infinitely many possible sets of initial states, and even for a single set of initial states, the set of reachable states is infinite. Due to these properties standard model checking techniques are not applicable for this kind of liveness problems.

Part II

Higher-Order Term Rewriting

Chapter 6

Introduction

In this chapter we introduce part II of this thesis. This part is based on the work on formalization of the higher-order version of the recursive path ordering (RPO), introduced in Section 1.3.4. We give a broad overview of this work in Section 6.1 and its motivation in Section 6.2. Section 6.3 describes main parts constituting this formalization and is followed by the short discussion of the history of this project, Section 6.4, and of related work, Section 6.5.

6.1 Overview

The recursive path ordering (RPO) goes back to Dershowitz [Der82]. It is a well-known reduction ordering for first-order term rewriting systems (TRSs). Jouannaud and Rubio generalized this ordering to the higher-order case thus creating the higher-order recursive path ordering (HORPO) [JR99, JR01, JR06, JR07].

This part of the thesis describes the formalization of this ordering with the proof of its well-foundedness. The first version of the ordering, as presented in [JR99], has been formalized along with some results from [JR01, JR07]. The variant of higher-order rewriting used for this work is the format of the algebraic-functional systems (AFSs), as introduced by Jouannaud and Okada [JO91].

The formalization has been carried out in the theorem prover Coq [The04, BC04], which is based on the calculus of inductive constructions. All the proofs in the formalization are constructive and make heavy use of dependent types, a feature provided by the intuitionistic type theory of Coq (version 8.1 of the prover has been used).

The higher-order terms used in this work were modeled by terms of the simply typed lambda calculus ($\lambda^{\rightarrow}$). Therefore part of the development, actually by far the biggest one, is devoted to the formalization of $\lambda^{\rightarrow}$, with termination of the β-reduction relation as a main result.

The multiset extension is used in HORPO for comparison of function arguments. Hence finite multisets and two variants of multiset extensions are part of this formalization.

Finally the last part of the formalization is the computability predicate proof method due to Tait and Girard [GTL89, Tai67], which is used in the proof of well-foundedness of HORPO.

This work is now part of the CoLoR [COL, BDCG$^+$06] library, the Coq library on rewriting and termination, which gathers results concerning termination of term rewriting, formalized in Coq (see Section 1.4 of this thesis for more details on CoLoR).

This part of the thesis contains an explanation of the theory being formalized as well as some remarks on the formalization, including Coq snippets. We tried to separate the two. The details of the formalization follow the explanation of the theory at the end of selected sections, denoted in the text as follows:

Coq ▸ *. . . some formalization details . . .* ◂

Hence the reader interested only in the content of the work and not in the way it has been carried out in Coq can safely skip the formalization details. Please note that the presentation of Coq scripts is slightly simplified for the sake of readability. The interested reader is encouraged to compare with the actual Coq scripts. Moreover the details of the Coq formalization are given only for the crucial definitions or to illustrate an interesting point. For a more detailed description we refer to [Kop06b].

6.2 Motivation

This work was motivated by the following concerns:

- **Verification of the proof.**
  Traditionally, trust in a mathematical proof is achieved by thorough checks done by the community. Complicated proofs that are little known inevitably contain small slips that remain unnoticed. Identification of such flaws and verification of the results justify the formalization effort. Indeed in the course of formalization we were able to detect a small flaw, concerning the use of multiset extension of an arbitrary relation, that could be easily repaired (we will discuss it shortly in Sections 7.2.1 and 9.3.3). In general [JR99, JR01] turned out to be a very favorable subject for formalization and the structure of the proofs could be followed to the letter in the formalization process.[1]

[1]Obviously providing formal proofs requires to be more explicit and to include all the results

- **Contribution to the CoLoR [COL, BDCG+06] project.**
  CoLoR is a project aiming at proving theoretical results from the area of term rewriting in the theorem prover Coq. Such theoretical results are used to transform termination proof candidates, produced by termination tools, into formal Coq proofs certifying termination. This development was contributed and became part of the CoLoR library.
- **Improving proof assistants technology.**
  In the future, technology will depend on critical systems. These systems will need to be proven correct before they are deployed. Theorem proving is still a rather laborious task and for the advancements in this area case studies are needed.

6.3 Contributions

The contribution is the Coq formalization of the proof of well-foundedness of HORPO [JR99]. It can be divided into the following parts:

- **Auxiliary results** (generally not discussed here)
  - Many operations and properties concerning lists and relations that were not present in the Coq standard library.
- **Multisets and multiset extensions of a relation** (Chapter 7).
  - Multisets as an abstract data-type (Section 7.1).
  - Concrete implementation of multisets using lists.
  - Definitions of two variants of a multiset extension of a relation (Section 7.2.1).
  - Multiset extensions preserve orderings (Section 7.2.3).
  - Multiset extensions preserve well-foundedness (Section 7.3).
- **Simply typed lambda calculus** (Chapter 8).
  - Definition of simply typed lambda terms over an arbitrary signature with constants and typing à la Church (Section 8.1).
  - Properties of environments, subterm relation and many further definitions and results (Section 8.2).
  - Typing properties: uniqueness of types, decidability of typing (Section 8.2.2).
  - Many-variable, typed substitution (Section 8.3).

that in normal presentation would be omitted as considered to be straightforward or irrelevant.

  - A convertibility relation on typed terms extending the concept of α-convertibility to free variables (Section 8.4).
  - β-reduction and its properties (Section 8.5).

- HORPO (Chapter 9).

  - Higher-order rewriting in the format of algebraic functional systems (AFSs) introduced in [JO91] (Section 9.1).
  - The computability predicate proof method by Tait and Girard and some computability properties used in the proof of well-foundedness of the union of HORPO and β-reduction (Section 9.2).
  - The definition of HORPO along the lines of [JR99] (Section 9.3.1).
  - Proofs of some properties of HORPO including its decidability (Section 9.3.2).
  - Proof of well-foundedness of the union of HORPO and β-reduction (Section 9.3.3).

Coq ▸ Figure 6.1 depicts relative sizes of those four parts measured by the size of Coq scripts; precise figures are given in Table 6.1. ◂

imgs/devsize.eps

Figure 6.1 Relative size of different parts of the Coq formalization.

Part of the development	No. of files	Lines of code	Total files size
Auxiliaries	6	2,780	70,455
Multisets	6	4,432	130,554
Simply typed lambda calculus	17	13,951	440,295
HORPO	4	3,027	100,239
TOTAL	33	24,190	741,543

Table 6.1 Size of different parts of the Coq formalization.

6.4 History

The development started in early 2004 as a Master Thesis [Kop04], supervised by Femke van Raamsdonk. After six months of full-time work it was completed and reported during the informal meeting of the Dutch Proof Tools day. The proofs were all finished, only the computability properties were assumed as axioms.

In the second phase of the project the goal was to prove all the computability properties. This turned out to be a very involved task and easily the most difficult part of the whole project. It required refinements to the theory developed so far and called for many new results. After completion the formalization tripled in size but was finally axiom-free. The part concerning HORPO (Section 9.3 of this thesis) was reported in [Kop06a] with the extended version available as a technical report [Kop06b].

6.5 Related Work

Simply typed lambda calculus and (first-order) RPO have been subject to many formalization efforts to date. However to the best of our knowledge our contribution is the first formalization of the higher-order variant of the recursive path ordering. Below we list a few somehow related formalizations. We begin with some formalizations of typed lambda calculi.

- Berger et al. in their recent work [BBLS06] proved strong normalization of $\lambda^{\rightarrow}$ in three different theorem provers, including Coq, and from those proofs machine-extracted implementations of normalization algorithms. Their formalization is closely related to our formalization of $\lambda^{\rightarrow}$. They used, just as we do, terms in de Bruijn notation and typing à la Church and their normalization proof also relies on Tait's computability predicate proof method, however their terms do not contain constants. The main difference between their formalization and the part of the formalization presented in Chapter 8 is the fact that their prime goal was extraction of a certified normalization algorithm, whereas for us a somewhat more complete formalization of $\lambda^{\rightarrow}$ was required with the application to HORPO in mind.

- Another source of formalizations of lambda calculi is the PoplMark challenge [ABF+05]: a set of benchmarks for measuring progress in the area of formalizing metatheory of programming languages. Among numerous submissions to PoplMark there are even few using Coq and de Bruijn representation of terms. The comparison however is difficult: although the benchmark is designed for a richer type system (System $F_{<:}$) it focuses on completely different aspects.

- Other formalizations include strong normalization proofs for calculi like: Calculus of Constructions [Bar99, Alt93b]; System F [Alt93a]; typed λ-calculus with co-products [ADHS01] and λ-calculi with weak reduction strategies [SDB06].

There are also some formalizations of RPO that are worth mentioning here.

- Murthy [Mur90] formalizes a classical proof of Higman's lemma, a specific instance of Kruskal's tree theorem, in a classical extension of Nuprl 3. The classical proof is due to Nash-Williams and uses a minimal bad sequence argument. The formalized classical proof was automatically translated into a constructive proof using Friedman's A-translation.

- Berghofer [Ber04] presents a constructive proof of Higman's lemma in Isabelle. The constructive proof is due to Coquand and Fridlender.

- Persson [Per99] presents a constructive proof of well-foundedness of a general form of recursive path relations. This proof is very similar to, and independently obtained, of the specialization to the first-order case of the proof of well-foundedness of HORPO by Jouannaud and Rubio [JR99]. The proof in [Per99] is extracted from the classical proof using a minimal bad sequence argument by using open induction due to Raoult [Rao88]. Persson presents an abstract formalization of well-foundedness of recursive path relations in the proof-checker Agda.

- Leclerc [Lec95] presents a formalization of well-foundedness of the multiset path ordering (MPO) in Coq. The focus is on giving upper bounds for descending sequences.

- De Kleijn in her Master Thesis [Kle03] shows well-foundedness of RPO in Coq. Her development however is not complete and contains a number of axioms.

- Coupet-Grimal and Delobel [CGD06] have provided a full development of well-foundedness of RPO in Coq. In their formalization they use multisets and a multiset extension of a relation that are part of the formalization which is the subject of this part of the thesis and which are described in Chapter 7.

The rest of this part of the thesis is organized as follows. In Chapter 7 finite multisets and two variants of a multiset extension of a relation are introduced. Chapter 8 presents simply typed lambda calculus. Chapter 9 introduces: higher-order rewriting, the computability predicate proof method and HORPO along with the proof of well-foundedness of its union with β-reduction of $\lambda^{\rightarrow}$.

Chapter 7

Finite Multisets

7.1 Multisets

Multisets extend the notion of a set by relaxing the condition that all elements are pairwise different. So in a multiset an element may occur a finite number of times and the number of its occurrences will be called a *multiplicity* of the element. Formally a multiset over a given domain A is represented by a function assigning a natural number (multiplicity) to every element of the domain. This function is a multiset counterpart of a characteristic function for sets. In the following we fix a set A.

Definition 7.1. [Multisets] A *multiset* M is a function $M : A \to \mathbb{N}$.

A *finite multiset* is a multiset for which there are only finitely many x such that $M(x) > 0$. We denote the set of finite multisets over A by $\mathbb{M}_A$. $\diamond$

In this work we focus on finite multisets only. Although some theory about multisets extends to infinite case, the parts we are interested in do not: only finite multisets can be treated as a data-type and the crucial property of a multiset extension of a relation, preservation of well-foundedness, does not hold if we allow multisets to contain infinitely many elements.

Coq ▸ We continue with showing how multisets can be defined in `Coq`. We use the `Coq` module mechanism to develop an abstract specification of multisets. We create a module type parameterized by a set A equipped with a decidable equality.[1] This module declares a `Multiset` data-type and the crucial function, `mult`, which given a multiset and an element returns the multiplicity of the given element in

[1] Decidability of equality will be required to prove decidability of the multiset extension (as we shall see in Theorem 7.7), which in turn will be needed to prove decidability of HORPO (see Theorem 9.22).

that multiset:

```
Parameter Multiset : Set.
Parameter mult : Multiset -> A -> nat.
```

Note that this is a very generic approach. The `mult` function is required, that for an arbitrary multiset must give a function from A to natural numbers, corresponding to the mathematical model of multisets as in Definition 7.1. But the `Multiset` type is not constrained in any way, allowing for any possible representation of this data-type.[2]

Further the specification calls for the existence of an empty multiset and of the following operations on multisets: equality, construction of a singleton multiset and union, intersection and difference of multisets. The summary of those operations with the corresponding Coq declarations is presented in Table 7.1. The specification of those operations is given in terms of the `mult` function and is presented in Table 7.2. For the detailed list of Coq statements expressing such specifications we refer to [Kop06b, p. 13].

Operation	Coq declaration	Coq notation
$M(x)$	`mult: Multiset -> A -> nat`	`x/M`
$M = N$	`meq: Multiset -> Multiset -> Prop`	`M =mul= N`
$\varnothing$	`empty: Multiset`	`empty`
$\{\{x\}\}$	`singleton: A -> Multiset`	`{{ x }}`
$M \cup N$	`union: Multiset -> Multiset -> Multiset`	`M + N`
$M \cap N$	`intersection: Multiset -> Multiset -> Multiset`	`M # N`
$M \backslash N$	`diff: Multiset -> Multiset -> Multiset`	`M - N`

Table 7.1 Primitive operations on multisets.

Note that those operations are not completely independent. For instance intersection can be defined using the difference operator as:

$$M \cap N := M \backslash (M \backslash N)$$

[2] Obvious representation as a function from A to $\mathbb{N}$ is just one of the possibilities. Note however that, if the domain A is infinite, with such representation enumerating all elements of a given multiset is impossible (just as in any functional programming language).

Operation	Specification
$M = N$	$M = N \iff \forall_x \; M(x) = N(x)$
$\varnothing$	$\varnothing(x) := 0$
$\{\{x\}\}$	$\{\{x\}\}(x) := 1$ $\{\{x\}\}(y) := 0 \quad x \neq y$
$M \cup N$	$(M \cup N)(x) := M(x) + N(x)$
$M \cap N$	$(M \cap N)(x) := min\{M(x), N(x)\}$
$M \backslash N$	$(M \backslash N)(x) := M(x) \dot{-} N(x)$ where: $m \dot{-} n := \begin{cases} m - n & \text{if } m \geqslant n \\ 0 & \text{otherwise} \end{cases}$

Table 7.2 Specification of primitive operations on multisets.

Both operations have been kept for efficiency reasons[3] but the above definition has been proven to realize the specification of the intersection operation.

One more additional requirement is the validity of the following inductive reasoning principle for multisets (P being an arbitrary proposition over multisets).

$$\frac{P \; \varnothing \qquad \forall_{M \in \mathbb{M}_A,\, a \in A} \; P \; M \implies P \; (M \cup \{\{a\}\})}{\forall_{M \in \mathbb{M}_A} \; P \; M}$$

So if we can prove that a property holds for an empty multiset and if assuming it holds for M we can prove that it also holds for $M \cup \{\{a\}\}$ (for an arbitrary a) then it holds for every multiset. It is stated in Coq as follows:

```
Axiom mset_ind_type: forall P : Multiset -> Type,
  P empty -> (forall M a, P M -> P (M + {{a}})) ->
  forall M, P M.
```

Using those primitives some additional operations are defined. For details we again refer to [Kop06b].

Then a simple list implementation of multisets has been provided. Multisets are represented as lists, the equality of multisets corresponds to the permutation predicate on lists, the union operation is realized by list concatenation and all other operations are implemented in a fairly straightforward way.

[3]It may be possible to provide a more efficient implementation for computing intersection of two multisets than by the above computation. Thanks to this choice our development of multisets can be reused also in a context where multisets are treated as a data-type and where efficiency is an issue.

Also a number of simple properties about multisets have been proven. They depend only on the axiomatic specification. This ensures that given another implementation of multisets, say, a more efficient one, those results carry over automatically. The same holds for all the rest of the multiset theory presented in the following sections so all the results are independent of the actual implementation of the multiset data-type. ◂

7.2 Multiset Extensions of a Relation

7.2.1 Definition

In this section we present two rather standard definitions (see for instance [BN98]) of an extension of a relation on elements to a relation on multisets of elements. It is usual to define those extensions for orderings, however we define them for arbitrary relations and only in Section 7.2.3 we study some properties valid only in case the underlying relation is an ordering. Having multiset extension of an arbitrary relation will be useful in Section 9.3.1 where we will use a multiset extension of HORPO, without proving its transitivity (more complex versions of HORPO are not even transitive).

The intuition behind the subsequent definition is as follows: to prove that a multiset M is bigger than N we are allowed to remove an arbitrary element a from M and any number of elements from N that are smaller than a. If by repetition of this process we can make those two multisets equal (in particular: empty) then we have proven that M is bigger than N. The single step of this process is captured in the following definition by the notion of the *multiset reduction* and since we may use more than one step the actual *multiset extension of a relation* is just a transitive closure of that multiset reduction.

Definition 7.2. [Multiset extension of a relation, $>_{mul}$] Let $>$ be a relation on A. We define the *multiset reduction* relation $\rhd_{mul}$ on $\mathbb{M}_A$ as:

$$M \rhd_{mul} N \text{ iff } \begin{cases} \exists_{X,Y \in \mathbb{M}_A; a \in A} \text{ such that:} \\ - \;\; M = X \cup \{\{a\}\} \\ - \;\; N = X \cup Y \\ - \;\; \forall_{y \in Y} \; a > y \end{cases}$$

We will say that a proves (or: is a witness for) $X \cup \{\{a\}\} = M \rhd_{mul} N = X \cup Y$.[4]

Now the *multiset extension of a relation* ($>_{mul}$) is defined to be the transitive closure of the multiset reduction, so $>_{mul} = \rhd^{+}_{mul}$. ◇

[4]Note that X and Y can be reconstructed as: $X = M \backslash \{\{a\}\}$, $Y = N \backslash X$.

Another way of defining a multiset extension is to combine together the "small steps" from the above definition and obtain a "big steps" variant as presented below.

Definition 7.3. [Multiset extension of a relation, $>_{MUL}$] Let $>$ be a relation on A. We define the multiset extension of a relation $>_{MUL}$ as a relation on $\mathbb{M}_A$:

$$M >_{MUL} N \text{ iff } \begin{cases} \exists_{X,Y,Z \in \mathbb{M}_A} \text{ such that:} \\ -\ Y \neq \varnothing \\ -\ M = X \cup Y \\ -\ N = X \cup Z \\ -\ \forall_{z \in Z}\ \exists_{y \in Y}\ y > z \end{cases}$$

We will say that Y proves (or: is a witness for) $X \cup Y = M >_{MUL} N = X \cup Z$. Sometimes for clarity we will mention the whole triple $\langle X, Y, Z \rangle$ as a witness for $M >_{MUL} N$. ⋄

First let us remark that for $>$ being an arbitrary relation $>_{mul}$ and $>_{MUL}$ do differ as illustrated on the following example.

Example 7.4. *Take* $A = \{a, b, c\}$ *and* $> = \{(a, b), (b, c)\}$*, then* $\{a\} >_{mul} \{c\}$ *because* $\{a\} \rhd_{mul} \{b\}$ *and* $\{b\} \rhd_{mul} \{c\}$*, but not* $\{a\} >_{MUL} \{c\}$*.* ◁

If $>$ is transitive (particularly, if it is an ordering) then the definitions of $>_{mul}$ and $>_{MUL}$ coincide as we will see in Section 7.2.3.

Coq ▸ Coq variants of Definitions 7.2 and 7.3 are straightforward; for details we refer to [Kop06b]. ◂

7.2.2 Properties of Multiset Extensions of a Relation

As remarked in the previous section $>_{mul}$ and $>_{MUL}$ differ for non-transitive relations. For now we will prove that $>_{MUL}$ is a subset of $>_{mul}$. We will use this result in Section 7.2.3 to prove equivalence of $>_{mul}$ and $>_{MUL}$ for orderings and also to conclude well-foundedness of $>_{MUL}$ from well-foundedness of $>_{mul}$ in Section 7.3. We begin with an auxiliary result.

Lemma 7.5. *Let* $X, Y \in \mathbb{M}_A$ *and* $a \in A$*. If* $\forall_{y \in Y}\ \exists_{x \in X \cup \{\{a\}\}}\ x > y$ *then there exist multisets* Y_X *and* Y_a *such that:*

- $Y = Y_a \cup Y_X$,
- $\forall_{y \in Y_a}\ a > y$,
- $\forall_{y \in Y_X}\ \exists_{x \in X}\ x > y$.

Proof. Easy induction on Y; for details see [Kop06b]. $\square$

Lemma 7.6. $\forall_{M,N \in \mathbb{M}_A}\ M >_{MUL} N \implies M >_{mul} N.$

Proof. We have $M = X \cup Y >_{MUL} X \cup Z = N$ with Y being the witness. We prove $X \cup Y >_{mul} X \cup Z$ by induction on the multiset Y. The base case is easily discarded as $Y \neq \varnothing$ by the definition of $>_{MUL}$.

For the induction step we have $X \cup Y \cup \{\{a\}\} >_{MUL} X \cup Z$ proven by $Y \cup \{\{a\}\}$ and we need to show $X \cup Y \cup \{\{a\}\} >_{mul} X \cup Z$. We distinguish two cases:

- $Y = \varnothing$. Then a proves $X \cup Y \cup \{\{a\}\} >_{mul} X \cup Z$.
- $Y \neq \varnothing$. By Lemma 7.5 we split Z into Z_a and Z_Y such that $Z = Z_a \cup Z_Y$, $\forall_{z \in Z_a}\ a > z$ and $\forall_{z \in Z_Y}\ \exists_{y \in Y}\ y > z$. Now we will continue with showing that $X \cup Y \cup \{\{a\}\} >_{mul} X \cup Z_Y \cup \{\{a\}\} >_{mul} X \cup Z$ which by transitivity of $>_{mul}$ will complete the proof.
  - $X \cup Y \cup \{\{a\}\} >_{mul} X \cup Z_Y \cup \{\{a\}\}$ by application of the induction hypothesis. For that we need to show $X \cup Y \cup \{\{a\}\} >_{MUL} X \cup Z_Y \cup \{\{a\}\}$ which is proven by Y as both $Y \neq \varnothing$ and $\forall_{z \in Z_Y}\ \exists_{y \in Y}\ y > z$ by assumption.
  - $X \cup Z_Y \cup \{\{a\}\} >_{mul} X \cup Z$ is easily proven by a. $\square$

Clearly $>_{MUL}$ is decidable provided we can decide $>$ as we will show in the following theorem. We will need this result to prove decidability of HORPO in Section 9.3.2.

Theorem 7.7. *If $>$ is a decidable relation then $>_{MUL}$ is also decidable.*

Proof. To decide whether $M >_{MUL} N$ we need to search for a witness Y. Since $Y \subseteq M$ there are only finitely many potential witnesses and we can consider all of them. (The Coq proof and, as a consequence, an algorithm, is slightly more involved. See notes on the Coq implementation below). $\square$

Let us conclude this section with a simple property of $>_{mul}$ and $>_{MUL}$ stating that every element in a smaller multiset is a reduct of some element in a bigger multiset.

Lemma 7.8.

(i) If $M >_{mul} N$ then $\forall_{n \in N}\ \exists_{m \in M}\ m >^* n$.

(ii) If $M >_{MUL} N$ then $\forall_{n \in N}\ \exists_{m \in M}\ m \geqslant n$.

Proof. (ii) is immediate from the definition of $>_{MUL}$. For (i) first let us note that if $M \rhd_{mul} N$ then $\forall_{n \in N}\ \exists_{m \in M}\ m \geqslant n$. The main goal easily follows as $>_{mul} = \rhd^*_{mul}$. $\square$

Note that the conclusion in case (i) of the above lemma being $m >^* n$ and not $m \geqslant n$ is essential and for non-transitive $>$ makes a difference. This property in this wrong form was (implicitly) stated in [JR99, (page 407, case 3 in the proof of Property 3.5)]. Although there it is only a very minor flaw that can be very easily repaired, it shows that one needs to be careful with reasoning about multiset extensions of non-transitive relations.

Coq ▸ We would like to make a short comment on the proof of decidability of the multiset extension $>_{MUL}$ (assuming decidability of $>$). Since the proof is constructive it provides a decision procedure for the problem: "given two multisets M and N does $M >_{MUL} N$ hold?". At the moment the used algorithm is suboptimal and it essentially considers all possible ways of splitting M into X and Y such that $X \subseteq N$ and Y is not empty and then for every element $z \in Z = N \backslash X$ looks for $y \in Y$ such that $y > z$ by examining all elements in Y. Note however that it would do to restrict X to the biggest set such that $X \subseteq M$ and $X \subseteq N$, so we can take $X = M \cap N$. Then $Y = M \backslash X$ and $Z = N \backslash X$. This observation leads to a more efficient decision procedure. The proof, however, will be more complex as we need to show that it is enough to consider only this single, potential witness. This improved procedure is not yet implemented in Coq. ◂

7.2.3 Multiset Extensions of Orderings

In this section we investigate properties of multiset extensions in case $>$ is an ordering. We will prove that then multiset extensions are also orderings and therefore in that context are often called *multiset orderings*. We will also show that $>_{mul}$ and $>_{MUL}$ coincide if $>$ is transitive (so in particular if it is an ordering).

We begin by proving transitivity of $>_{MUL}$. For that we first introduce two auxiliary lemmas.

Lemma 7.9. $\forall_{l,r,u,d \in \mathbb{N}}\ l + r = u + d \implies r = \min\{r, d\} + (u \dot{-} l)$

Proof. By case analysis with inequalities $u \geqslant l$ and $r \geqslant d$ using definitions of min and $\dot{-}$. □

Lemma 7.10. $\forall_{L,R,U,D \in \mathbb{M}_A}\ L \cup R = U \cup D \implies R = (R \cap D) \cup (U \backslash L)$.

Proof. Immediate from 7.9; for an alternative direct proof, see [Kop06b]. □

Lemma 7.11. *Let $>$ be a transitive relation. Then $>_{MUL}$ is also transitive.*

Proof. We have $M >_{MUL} N >_{MUL} P$ with $\langle X_1, Y_1, Z_1\rangle$ proving $M >_{MUL} N$ and $\langle X_2, Y_2, Z_2\rangle$ proving $N >_{MUL} P$. We claim that then $\langle X_1 \cap X_2, Y_1 \cup (Y_2 \backslash Z_1), Z_2 \cup (Z_1 \backslash Y_2)\rangle$ proves $M >_{MUL} P$ as:

- $Y_1 \cup (Y_2 \backslash Z_1) \neq \varnothing$ as $Y_1 \neq \varnothing$.
- $M = X_1 \cup Y_1 = (X_1 \cap X_2) \cup (Y_2 \backslash Z_1) \cup Y_1$. First equality is due to $M >_{MUL} N$ with $\langle X_1, Y_1, Z_1 \rangle$. The second one is by Lemma 7.10 as $X_1 \cup Z_1 = N = X_2 \cup Y_2$.
- $N = X_1 \cup Z_1 = (X_1 \cap X_2) \cup (Z_1 \backslash Y_2) \cup Z_2$. Analogously to the above case. □

Now we can prove equivalence of $>_{mul}$ and $>_{MUL}$ for transitive relations.

Theorem 7.12. *Let $>$ be transitive then* $\forall_{M,N \in \mathbb{M}_A}\ M >_{mul} N \iff M >_{MUL} N$.

Proof. ($\Rightarrow$) Induction on $M >_{mul} N$. Either $M \rhd_{mul} N$ with a but then $M >_{MUL} N$ with $\{\{a\}\}$; or $M >_{mul} M' >_{mul} N$ but then $M >_{MUL} M' >_{MUL} N$ by the induction hypothesis and we conclude by transitivity of $>_{MUL}$ (Lemma 7.11; note the use of transitivity of $>$ here).

($\Leftarrow$) By Lemma 7.6. □

Now we continue with showing that if $>$ is a strict ordering then so are $>_{MUL}$ and $>_{mul}$. We need an auxiliary lemma first.

Lemma 7.13. $\forall_{M,N,P \in \mathbb{M}_A}\ M >_{MUL} N \iff M \cup P >_{MUL} N \cup P$

Proof. Observe that for arbitrary $R, S \in \mathbb{M}_A$, if $R >_{MUL} S$ then it is proven by $R \cap S$. Hence $M >_{MUL} N$ is proven by $M \cap N$ iff $M \cup P >_{MUL} N \cup P$ is proven by $P \cup (M \cap N)$. □

Theorem 7.14. *If $>$ is a strict ordering, then $>_{MUL}$ and $>_{mul}$ are also strict orderings.*

Proof. By Theorem 7.12 for orderings $>_{MUL} = >_{mul}$. We continue by proving that $>_{MUL}$ is a strict ordering. We get transitivity by Lemma 7.11. For irreflexivity, by Lemma 7.13, we get $M >_{MUL} M \implies \varnothing >_{MUL} \varnothing$ but by definition $\varnothing$ is a minimal element of $>_{MUL}$. □

7.3 Well-foundedness of Multiset Extensions of a Relation

In this section we will prove that the two variants of multiset extension introduced before preserve well-foundedness, so a multiset extension of a well-founded relation is again well-founded. The proof that we give is due to Buchholz and closely follows the presentation by Nipkow [Nip98]. It proceeds by well-founded part induction, see Definition 1.8).

We will abbreviate $\mathbb{M}_A{}^{Acc}_{\lhd_{mul}}$ by $\mathbb{M}^{Acc}_{\lhd}$. By $\lhd_{mul}$ we denote the converse of $\rhd_{mul}$; similarly for $<_{mul}$ and $<_{MUL}$. We begin with a simple auxiliary lemma considering $\lhd_{mul}$.

Lemma 7.15. *Let $M, N \in \mathbb{M}_A$. If $N \lhd_{mul} M \cup \{\{a\}\}$ then there exist a multiset $M' \in \mathbb{M}_A$ such that:*

$$\begin{pmatrix} N = M' \cup \{\{a\}\} \\ M' \lhd_{mul} M \end{pmatrix} \vee \begin{pmatrix} N = M \cup M' \\ \forall_{x \in M'}\ x < a \end{pmatrix}$$

Proof. Easy from the definition of $\lhd_{mul}$ (see [Kop04] for a detailed proof). □

Lemma 7.16. *Let $M_0 \in \mathbb{M}^{Acc}_{\lhd}$ and $a \in A$. Then:*

$$\left.\begin{array}{ll} (1) & \forall_{b<a,\ M \in \mathbb{M}^{Acc}_{\lhd}}\ M \cup \{\{b\}\} \in \mathbb{M}^{Acc}_{\lhd} \\ (2) & \forall_{M \lhd_{mul} M_0}\ M \cup \{\{a\}\} \in \mathbb{M}^{Acc}_{\lhd} \end{array}\right\} \implies M_0 \cup \{\{a\}\} \in \mathbb{M}^{Acc}_{\lhd}$$

Proof. By the definition of $\mathbb{M}^{Acc}_{\lhd}$ we need to show $N \in \mathbb{M}^{Acc}_{\lhd}$ for $N \lhd_{mul} M_0 \cup \{\{a\}\}$. By Lemma 7.15 there are two possibilities for $N \lhd_{mul} M_0 \cup \{\{a\}\}$:

- $N = M \cup \{\{a\}\}$ for some $M \lhd_{mul} M_0$. Then $N \in \mathbb{M}^{Acc}_{\lhd}$ by (2).
- $N = M_0 \cup K$ and $\forall_{k \in K}\ k < a$. We proceed by induction on K. For base case if $K = \varnothing$ then $N = M_0 \in \mathbb{M}^{Acc}_{\lhd}$ by assumption. For induction step $K = K_0 \cup \{\{k\}\}$ and $K_0 \in \mathbb{M}^{Acc}_{\lhd}$ by the induction hypothesis. So $N \in \mathbb{M}^{Acc}_{\lhd}$ follows from (1) as $k < a$. □

Lemma 7.17. $\forall_{b<a,\ M \in \mathbb{M}^{Acc}_{\lhd}}\ M \cup \{\{b\}\} \in \mathbb{M}^{Acc}_{\lhd} \implies \forall_{M \in \mathbb{M}^{Acc}_{\lhd}}\ M \cup \{\{a\}\} \in \mathbb{M}^{Acc}_{\lhd}$

Proof. Follows immediately from Lemma 7.16 by well-founded part induction on M. □

Lemma 7.18. $\forall_{a \in A^{Acc}_{<},\ M \in \mathbb{M}^{Acc}_{\lhd}}\ M \cup \{\{a\}\} \in \mathbb{M}^{Acc}_{\lhd}$

Proof. Follows immediately from Lemma 7.17 by well-founded induction on a. □

Lemma 7.19. $\forall_{M \in \mathbb{M}_A}\ M \in \mathbb{M}^{Acc}_{\lhd}$

Proof. By induction on M. For base case $\varnothing \in \mathbb{M}^{Acc}_{\lhd}$ as there is no multiset N such that $N \lhd_{mul} \varnothing$. The induction step follows from Lemma 7.18. □

Theorem 7.20. *If $<$ is a well-founded relation then its multiset extensions $<_{mul}$ and $<_{MUL}$ are also well-founded.*

Proof. $<_{mul}$ is well-founded as it is a transitive closure of $\lhd_{mul}$ which is well-founded by Lemma 7.19. Then well-foundedness of $<_{MUL}$ follows from Lemma 7.6. □

Coq ▸ Nipkow remarked that the variant of a proof of well-foundedness of the multiset extension from [Nip98] is particularly well suited for theorem provers. Indeed this proof went quite smoothly in Coq and it is rather short. ◂

Chapter 8

Simply Typed λ-calculus

In this chapter we will present the simply typed lambda calculus ($\lambda^{\rightarrow}$) that was introduced by Church in 1940 [Chu40]. For a more detailed introduction of $\lambda^{\rightarrow}$we refer to, for instance, [Bar92, Chu40]. For some standard and/or easy theorems we omit the proofs, but they are clearly part of the formalization and they can be found in [Kop06b].

We will define terms in Section 8.1, then we will introduce some further definitions and results in Section 8.2. Section 8.3 will be devoted to the definition of a many-variable, typed substitution and Section 8.4 to the development of an equivalence relation on terms that extends the concept of α-convertibility to free variables. Finally Section 8.5 introduces the β-reduction relation along with its properties.

8.1 Terms

We assume a set of sorts (ground types) and we define simple types.

Definition 8.1. [Simple types, $\mathcal{T}_{\mathcal{S}}$] Assume a set of *sorts* $\mathcal{S}$. We inductively define a set of *simple types* $\mathcal{T}_{\mathcal{S}}$ as follows:

- for any $\sigma \in \mathcal{S}$, $\sigma \in \mathcal{T}_{\mathcal{S}}$ *(base type)*,
- if $\sigma, \tau \in \mathcal{T}_{\mathcal{S}}$ then $\sigma \rightarrow \tau \in \mathcal{T}_{\mathcal{S}}$ *(arrow type)*.

We will denote simple types by σ, τ, ρ. $\diamond$

Now we can define a notion of a signature.

Definition 8.2. [Signature, Σ] We assume a set of *constants* $\mathcal{F}$ and we define a *signature* (Σ) to be a set of typed constants declarations, that is pairs $f:\sigma$ with $f \in \mathcal{F}$ and $\sigma \in \mathcal{T_S}$.

We will usually refer to typed constants as *function symbols* and we will denote them by f, g etc. $\diamond$

For the rest of the presentation we assume a set of sorts $\mathcal{S}$ and a signature Σ to be fixed and we assume a fixed set of variables $\mathcal{V}$, disjoint from Σ.

We define environments to hold declarations for free variables.

Definition 8.3. [Environment, $\mathcal{E}nv$] The *environment* ($\mathcal{E}nv$) is defined as a finite set of distinct variable declarations, that is: $\mathcal{E}nv \subset \mathcal{V} \times \mathcal{T_S}$ such that for every environment $\Gamma : \mathcal{E}nv$, $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ for all $1 \leqslant i, j \leqslant n$: $x_i \neq x_j$ for $i \neq j$. The domain of the environment is defined as $\mathit{Var}(\Gamma) = \{x_1, \ldots, x_n\}$.

We will denote environments as Γ, Δ etc. $\diamond$

Now we define un-typed terms.

Definition 8.4. [Un-typed terms, $\mathcal{P}t$] A set of *un-typed terms* is defined by the following grammar:

$$\mathcal{P}t \ := \ \mathcal{V} \mid \Sigma \mid @(\mathcal{P}t, \mathcal{P}t) \mid \lambda\mathcal{V}:\mathcal{T_S}.\mathcal{P}t \qquad \diamond$$

The grammar rules for terms define respectively: a variable, a function symbol, an application and an abstraction. Application is left-associative and we will write $@(t, u, s)$ for $@(@(t, u), s)$. Also an application headed by a function symbol we will write as $f(t_1, \ldots, t_n)$ abbreviating $@(f, t_1, \ldots, t_n)$.

Now we proceed with presenting typing judgements: a typing discipline that our typed terms will follow.

Definition 8.5. [Typing judgements] We will write *typing judgements* of the form $\Gamma \vdash t : A$ to denote that in an environment Γ a term t has type A. They respect the rules of the following inference system:

$$\frac{x:\sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad\qquad \frac{f:\sigma \in \Sigma}{\Gamma \vdash f : \sigma}$$

$$\frac{\Gamma \vdash t : \sigma \to \tau \qquad \Gamma \vdash u : \sigma}{\Gamma \vdash @(t, u) : \tau} \qquad\qquad \frac{\Gamma \cup \{x:\sigma\} \vdash t : \tau}{\Gamma \vdash \lambda x:\sigma.t : \sigma \to \tau} \qquad \diamond$$

Definition 8.6. [Typed terms, Λ] *Typed terms* (Λ) are triples (Γ, t, σ) consisting of an environment Γ, an un-typed term t and a type σ, such that $\Gamma \vdash t : \sigma$ holds.

Such triples will be identified with their corresponding typing judgements. We also define:

$$\text{env}(\Gamma \vdash t : \sigma) = \Gamma, \quad \text{term}(\Gamma \vdash t : \sigma) = t, \quad \text{type}(\Gamma \vdash t : \sigma) = \sigma$$

We will denote typed terms by letters t, u etc. Often we will omit the environments and write $t{:}\sigma$ instead of $\Gamma \vdash t : \sigma$ or even only t if the type is irrelevant. ⋄

Now we will define fully applied terms, that is terms where function symbols are applied to all its arguments.

Definition 8.7. [Fully applied terms, $\overline{\Lambda}$] *Fully applied terms* ($\overline{\Lambda}$) are typed terms that respect the following inference rules, where we speak about typed terms but for readability omit environments and types where not needed:

$$\frac{}{x \in \overline{\Lambda}} \qquad \frac{t \in \overline{\Lambda}}{\lambda x{:}\sigma.t \in \overline{\Lambda}}$$

$$\frac{t \in \overline{\Lambda} \qquad u \in \overline{\Lambda}}{@(t, u) \in \overline{\Lambda}} \qquad \frac{t_1 \in \overline{\Lambda}, \ldots, t_n \in \overline{\Lambda}, \sigma \in \mathcal{S}}{f(t_1, \ldots, t_n) : \sigma \in \overline{\Lambda}}$$

So for fully applied terms we admit all variables, we push the property through abstraction and application and, in the last rule, for an application headed by a function symbol, we demand full application resulting in a base type of a term. ⋄

Coq ▸ The development of $\lambda^{\rightarrow}$ in Coq is structured using Coq's module mechanism. Firstly in the file TermsSig.v the module type SimpleTypes is defined containing definition of simple types parameterized by the set of ground types. Then the module Signature, representing a signature, contains definition of constants along with the function `f_type` mapping them to their types. All further development concerning $\lambda^{\rightarrow}$ is done within functors taking such signature as their argument. In the file TermsDef.v the definition of typed terms is given.

Note that we use de Bruijn indices [Bru72] to represent terms in order to avoid having to explicitly deal with α-conversion. Due to that fact the environments could simply be represented by lists of simple types. However, later on we will introduce a lifting operation on terms that renames variables, which in case of de Bruijn indices corresponds to increasing their numerical values. This leaves some variables with lower indexes undeclared and to express that fact we need dummy variables. So environments are lists of SimpleType option[1], and not SimpleType, with None representing dummy variables. This will lead to some small complications as we will explain in Section 8.2.1. We will also use the notation

[1]Where `option` is a standard type in Coq, with `option A` being a type constructed by either providing a value x of type A (`Some x`) or indicating absence of such value (`None`); this can be thought of as a type-safe way of introducing null values, known from most programming languages.

`E |= x := A` to denote that the variable `x` in the environment `E` has type `A` and `E |= x :!` to denote that the variable `x` is undeclared in the environment `E` and `A --> B` for the type $A \to B$.

We continue with a straightforward definition of un-typed terms. Note that variables are natural numbers representing their index in de Bruijn notation. We introduce a number of notational conventions to make representation of terms more readable.

```
Inductive Preterm : Set :=
  | Var (x: nat)
  | Fun (f: FunctionSymbol)
  | Abs (A: SimpleType)(M: Preterm)
  | App (M N: Preterm).

Notation "^ f" := (Fun f) (at level 20).
Notation "% x" := (Var x) (at level 20).
Infix "@@" := App (at level 25, left associativity).
Notation "s [ x ]" := (s @@ x) (at level 30).
Notation "s [ x, y ]" := (s @@ x @@ y) (at level 30).
Notation "\ A => M" := (Abs A M) (at level 35).
```

Next we present typing judgements. They will be written in Coq in the form `E |- M := A` representing typing judgement $E \vdash M : A$. It is easy to recognize the inference system from the Definition 8.5 in the following inductive definition, where `A [#] E` is the environment `E` extended with a declaration of `A`.

```
Reserved Notation "E |- Pt := A" (at level 60).
Inductive Typing : Env -> Preterm -> SimpleType -> Set :=
| TVar: forall E x A,
          E |=  x := A ->
          E |- %x := A ->
| TFun: forall E f,
          E |- ^f := f_type f
| TAbs: forall E A B Pt,
          A [#] E |- Pt := B ->
          E |- \A => Pt := A --> B
| TApp: forall E A B PtL PtR,
          E |- PtL := A --> B ->
          E |- PtR := A ->
          E |- PtL @@ PtR := B
where
  "E |- Pt := A" := (Typing E Pt A).
```

Finally we give the definition of a typed term.

```
Record Term : Set := buildT {
```

```
  env:    Env;
  term:   Preterm;
  type:   SimleType;
  typing: Typing env term type
}.
```

Definitions introduced in this section are very crucial as we will constantly work with them while formalizing the content of the following sections. One may wonder whether it would not be more convenient to use a single dependent inductive definition of typed terms that would combine the definition of term structure with its typing judgement and that could look as follows:

```
Inductive Term : Env -> SimpleType -> Set :=
| TVar: forall E x A,
          E |= x := A ->
          Term E A
| TFun: forall E f,
          Term E (f_type f)
| TAbs: forall E A B,
          Term (A [#] E) B ->
          Term E (A --> B)
| TApp: forall E A B,
          Term E (A --> B) ->
          Term E A ->
          Term E B.
```

At first sight this definition looks very attractive but although having some advantages, it also has a serious drawback: the structure of terms is embedded within its typing judgement. As we shall see later the great part of the proofs of equality of two terms will use the observation, that is to be proven in Section 8.2.2, that two terms with equal structure and equal environments are equal. Such proofs essentially split the reasoning into the reasoning about term structure (un-typed λ-terms) and about environments. This is very convenient to do with the use of the first proposed definition as it requires only dealing with two very simple definitions (Env and Preterm) as opposed to the second approach where one needs to constantly work with a complex dependent type. Moreover in this way all the reasoning about term structures gives us some results about the theory of un-typed λ-calculus. ◂

8.2 Further Properties and Definitions of $\lambda^{\rightarrow}$

8.2.1 Environment Properties

We start with some simple operations on environments:

Definition 8.8. [Environment operations] For any environments Γ, Δ we define the binary operations of composition and subtraction of environments.

- $\Gamma \cdot \Delta = \Delta \cup \{x : \sigma \in \Gamma \mid x \notin \mathit{Var}(\Delta)\}$
- $\Gamma \backslash \Delta = \{x : \sigma \in \Gamma \mid x \notin \mathit{Var}(\Delta)\}$ ◇

We also introduce a notion of compatibility of environments:[2]

Definition 8.9. [Environment compatibility] For environments Γ, Δ we say that they are compatible iff for any variable v if they both declare it, they declare it with the same type. We will denote the fact that Γ is compatible with Δ by $\Gamma \rightleftharpoons \Delta$. So we have:

$$\Gamma \rightleftharpoons \Delta \;\equiv\; \forall_{x \in \mathcal{V},\, \sigma,\tau \in \mathcal{T_S}}\; x : \sigma \in \Gamma \wedge x : \tau \in \Delta \implies \sigma = \tau$$ ◇

Coq ▸ All the definitions and results concerning environments can be located in the file TermsEnv.v. Here we skip the definitions which are rather standard; for details we refer to [Kop06b].

We already indicated in Section 8.1, while introducing terms, that allowing dummy variables in environments (which will be useful in Section 8.3, in particular in 8.3.2) leads to problems. Those problems come from the fact that in this way we loose unique representation of environment. For instance, the empty environment can be represented by the empty list (`nil`) but also by a list with only a single declaration for dummy (`None::nil`). This problem was solved by providing equality for environments, different than Coq's Leibniz' equality, that takes those subtle representation issues into account. It is defined via a subset predicate for environments, that is $\Gamma = \Delta \iff \Gamma \subseteq \Delta \wedge \Delta \subseteq \Gamma$.

```
Definition envSubset E F :=
  forall x A, E |= x := A -> F |= x := A.
Definition env_eq E1 E2 := envSubset E1 E2 /\ envSubset E2 E1.
Notation "E1 [=] E2" := (env_eq E1 E2) (at level 70).
```
◂

8.2.2 Typing Properties

Now we will look into some properties of the type system of the simply typed λ-calculus. The first two properties ensure that every term has a unique type and that type derivations are unique.

Theorem 8.10 (Uniqueness of types)**.** *Suppose* $\Gamma \vdash t : \sigma$ *and* $\Gamma \vdash t : \tau$ *then* $\sigma = \tau$. □

[2] Hindley [Hin97] uses the term *consistence* for this concept but we follow the naming convention of Jouannaud and Rubio [JR07].

Theorem 8.11 (Uniqueness of typing judgements)**.** *Given a typed term $\Gamma \vdash t : \sigma$ its type derivation is unique.* □

Now we present a theorem stating that typability of simply typed λ-terms is decidable in linear time.

Theorem 8.12 (Decidability of typing)**.** *Given environment Γ and an un-typed term t the problem of finding σ such that $\Gamma \vdash t : \sigma$ is decidable in linear time with respect to the sum of the sizes of t and Γ.* □

Two following lemmas express the fact that a term can be typed in an extended environment. We will need them in Section 8.3.3 for reasoning about substitution.

Lemma 8.13. *Let Δ, Γ be an environment. If $\Gamma \vdash t : \sigma$ then $\Delta \cdot \Gamma \vdash t : \sigma$.*

Proof. Easy structural induction on t. □

Lemma 8.14. *Let Δ, Γ be environments. If $\Gamma \vdash t : \sigma$ and $\Delta \rightleftharpoons \Gamma$, then $\Gamma \cdot \Delta \vdash t : \sigma$.*

Proof. Easy structural induction on t. For the variable case we use compatibility of Δ with Γ. □

Coq ▸ Remarks:

- The proof of Theorem 8.11 is actually quite technical in Coq. It involves the usage of the uniqueness of identity proofs for dependent types and makes use of Streicher's K property from the standard library. I would like to express my gratitude to Roland Zumkeller who helped me carry out this proof.
- From the constructive Coq proof of Theorem 8.12 a certified algorithm for typing lambda terms can be extracted easily. This algorithm indeed has linear complexity, but note that linear complexity of the decision procedure is not proven in Coq. ◂

8.2.3 Further Definitions

We define a set of free variables of a term as:

Definition 8.15. [Free variables, Vars] For $\Gamma \vdash t : \sigma$ we define the environment containing free variables of t, $\mathrm{Vars}(t)$, by induction on t as:

$$\begin{array}{rcl}
\mathrm{Vars}(\Gamma \vdash x : \sigma) & = & \{x : \sigma\} \\
\mathrm{Vars}(\Gamma \vdash f : \sigma) & = & \varnothing \\
\mathrm{Vars}(\Gamma \vdash @(t_l, t_r) : \sigma) & = & \mathrm{Vars}(\Gamma \vdash t_l : \tau \to \sigma) \cdot \mathrm{Vars}(\Gamma \vdash t_r : \tau) \\
\mathrm{Vars}(\Gamma \vdash \lambda x : \sigma . t : \sigma \to \tau) & = & \mathrm{Vars}(\Gamma \cdot \{x : \sigma\} \vdash t : \tau) \backslash \{x : \sigma\}
\end{array}$$

Note that $\mathrm{Vars}(\Gamma \vdash t : \sigma) \subseteq \Gamma$. ◇

We define a subterm relation as follows:

Definition 8.16. [Sub-term, ⊏, ⊑] The *subterm* (⊑) and *strict subterm* (⊏) relations are inductively defined as:

$$\frac{t \sqsubseteq u_l}{t \sqsubset @(u_l, u_r)} \qquad \frac{t = u}{t \sqsubseteq u} \qquad \frac{t \sqsubseteq u_r}{t \sqsubset @(u_l, u_r)} \qquad \frac{t \sqsubset u}{t \sqsubseteq u} \qquad \frac{t \sqsubseteq u}{t \sqsubset \lambda x : \sigma . u}$$

◇

We proceed with showing that the strict subterm relation is well-founded. This result justifies the use of induction with respect to the ⊏ relation on terms. We will refer to it as an induction on the structure of terms and it will be frequently used in the subsequent proofs.

Theorem 8.17 (Well-foundedness of ⊏)**.** ⊏ *is a well-founded relation.* □

The following two notions will be used in the definition of HORPO in Section 9.3.

Definition 8.18. [Partial left-flattening] Given a term $@(t_1, \ldots, t_n)$, any list of terms of the form: $@(t_1, t_2, \ldots, t_i), t_{i+1}, \ldots, t_n$ for $1 \leqslant i \leqslant n$ is called its *partial left-flattening*. ◇

Definition 8.19. [Neutral term] A term t is called *neutral* if it is not an abstraction. ◇

Coq ▸ In Coq development an environment containing only free variables and no additional unused declarations is called an active environment of a term. It is defined as `activeEnv (M: Term) : Env`. Then a number of auxiliary functions is provided. Functions `appBodyL`, `appBodyR` and `absBody` return the left argument of an application, the right argument of an application and the body of an abstraction respectively and are defined as expected. Predicates `isVar`, `isFunS`, `isAbs` and `isApp` hold for a term that is a variable, a function symbol, an abstraction or an application respectively. For an application $t = @(t_1, t_2, \ldots, t_n)$ we call t_1 an application head, $t_2, \ldots, t_n$ application arguments and $t_1, \ldots, t_n$ application units. `appHead t` returns the head of t; `isArg t' t` (resp. `isAppUnit t' t`) holds if t' is an application argument of `t` (resp. application unit). Finally `subterm` and `subterm_le` correspond to the definitions of ⊏ and ⊑, respectively. ◂

8.3 Substitution

In this section we introduce the substitution on terms of $\lambda^{\rightarrow}$. First in 8.3.1 we introduce concepts not directly related to substitution, namely those of positions in a term and the replacement operation. In 8.3.2 we define lifting and lowering operations on terms, which will be used in the definition of substitution. Finally in 8.3.3 we define substitution on $\lambda^{\rightarrow}$ terms and discuss some of its properties.

8.3.1 Positions and Replacement

We begin by defining term positions.

Definition 8.20. [Term positions, $\mathcal{P}os_{\Lambda}$] We define *positions* ($\mathcal{P}os$) as strings over the following set:

$$\{\epsilon, \lhd, \rhd, \lambda\}$$

its elements indicating a position at the root, a position in left and right argument of an application and a position within a lambda abstraction respectively.

Now we inductively define *term positions* ($\mathcal{P}os_{\Lambda}$) as a family of positions indexed by terms, as follows:

$$\frac{t \in \Lambda}{\epsilon \in \mathcal{P}os_t} \qquad \frac{p \in \mathcal{P}os_t}{\lambda \cdot p \in \mathcal{P}os_{\lambda x:\sigma.t}}$$

$$\frac{p \in \mathcal{P}os_{t_l}}{\lhd \cdot p \in \mathcal{P}os_{@(t_l,t_r)}} \qquad \frac{p \in \mathcal{P}os_{t_r}}{\rhd \cdot p \in \mathcal{P}os_{@(t_l,t_r)}} \qquad \diamond$$

We continue with a definition of a subterm at a position and of a replacement of a term at a given position.

Definition 8.21. [Subterm at position, $t|_p$] For any term $\Gamma \vdash t : \sigma$ and position $p \in \mathcal{P}os_t$ we give a recursive definition of *a subterm of t at p, $t|_p$*:

$$\begin{aligned} t|_{\epsilon} &= t \\ @(t_l, t_r)|_{\lhd \cdot p} &= t_l|_p \\ @(t_l, t_r)|_{\rhd \cdot p} &= t_r|_p \\ \lambda x\!:\!\sigma.t|_{\lambda \cdot p} &= t|_p \end{aligned} \qquad \diamond$$

Definition 8.22. [Replacement at position, $t[u]_p$] For any term $\Gamma \vdash t : \sigma$, position $p \in \mathcal{P}os_t$ and term $\Delta \vdash u : \tau$ such that $\mathrm{type}(t|_p) = \tau$ and $\mathrm{env}(t|_p) = \Delta$ we define the *replacement in term t at position p with term u* ($t[u]_p$), by recursion on p, as:

$$\begin{aligned} t[u]_{\epsilon} &= u \\ @(t_l, t_r)[u]_{\lhd \cdot p} &= @(t_l[u]_p, t_r) \\ @(t_l, t_r)[u]_{\rhd \cdot p} &= @(t_l, t_r[u]_p) \\ \lambda x\!:\!\sigma.t[u]_{\lambda \cdot p} &= \lambda x\!:\!\sigma.t[u]_p \end{aligned} \qquad \diamond$$

Lemma 8.23. *For any terms $\Gamma \vdash t : \sigma$, $\Delta \vdash u : \tau$ and position $p \in \mathcal{P}os_t$ such that the replacement $t[u]_p$ is well defined we have:*

$$\Gamma \vdash t[u]_p : \sigma$$

so the result of a replacement is typable with the same environment and type as the term in which the replacement takes place.

Proof. Induction on p.

- $p = \epsilon$. Then $t[u]_p = u$, $\Gamma = \mathrm{env}(t|_\epsilon) = \mathrm{env}(t) = \Delta$ and $\tau = \mathrm{type}(t|_\epsilon) = \mathrm{type}(t) = \sigma$ by definition and then $\Gamma \vdash t[u]_p : \sigma$ as $\Delta \vdash u : \tau$.
- $p = \lambda \cdot p'$, then $t = \lambda x{:}\sigma.t'$ and $t[u]_p = \lambda x{:}\sigma.t'[u]_{p'}$ and we conclude by the induction hypothesis for $t'[u]_{p'}$.
- $p = \lhd \cdot p'$, then $t = @(t_l, t_r)$ and $t[u]_p = @(t_l[u]_{p'}, t_r)$ and we conclude by the induction hypothesis for $t_l[u]_{p'}$.
- $p = \rhd \cdot p'$, then $t = @(t_l, t_r)$ and $t[u]_p = @(t_l, t_r[u]_{p'})$ and we conclude by induction hypothesis for $t_r[u]_{p'}$. ☐

8.3.2 Lifting and Lowering of Terms

So far we tried to hide the use of de Bruijn indices from the presentation but it is not possible in this section as lifting and lowering are operations specifically defined for terms using this representation. So throughout this section we assume $\mathcal{V} = \mathbb{N}$.

Before we define those operations let us briefly explain why do we need them. In the process of substitution we replace a term in some context which may contain binders. To avoid capturing of free variables their de Bruijn indices need to be increased by a value equal to the number of binders in the context in which a substitution takes place. This operation is called a lifting of a term. Lowering is the opposite operation in which the indices are decreased.

Definition 8.24. [Term lifting, $t\!\uparrow^n_k$] For a term t and $n, k \in \mathbb{N}$ we define its lifted version with variables with index less than k untouched and remaining ones increased by n ($t\!\uparrow^n_k$) as:

$$\begin{aligned}
f\!\uparrow^n_k &= f \\
x\!\uparrow^n_k &= x && \text{if } x < k \\
x\!\uparrow^n_k &= x + n && \text{if } x \geqslant k \\
@(t_l, t_r)\!\uparrow^n_k &= @(t_l\!\uparrow^n_k, t_r\!\uparrow^n_k) \\
\lambda x{:}\sigma.t\!\uparrow^n_k &= \lambda x{:}\sigma.t\!\uparrow^n_{k+1}
\end{aligned}$$

We also define $t\!\uparrow^n := t\!\uparrow^n_0$. ⋄

Now we define lifting of environments:

Definition 8.25. [Environment lifting, $\Gamma\uparrow^n_k$] For an environment $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ we define its lifted version $\Gamma\uparrow^n_k$ as:

$$\Gamma\uparrow^n_k := \{x_i : \sigma_i \mid x_i : \sigma_i \in \Gamma, k > i \in \mathbb{N}\} \cdot \{(x_i + n) : \sigma_i \mid x_i : \sigma_i \in \Gamma, k \leqslant i \in \mathbb{N}\}$$

We also define $\Gamma\uparrow^n := \Gamma\uparrow^n_0$. $\diamond$

The following result ensures that lifted terms are well-typed.

Lemma 8.26. *If* $\Gamma \vdash t : \sigma$ *then for any* $n, k \in \mathbb{N}$*:* $\Gamma\uparrow^n_k \vdash t\uparrow^n_k : \sigma$.

Proof. Structural induction on t:

- $t = x$. Either $x < k$ or $x \geqslant k$ but in both cases $\Gamma\uparrow^x_k \vdash x\uparrow^n_k : \sigma$.
- $t = f$. Then $f\uparrow^n_k = f$ and $\Gamma\uparrow^n_k \vdash f\uparrow^n_k : \sigma$ by typing rule for constant.
- $t = \lambda x : \tau.t_b$ with $t_b : \xi$ and $\sigma = \tau \to \xi$. We conclude $\Gamma\uparrow^n_k \vdash (\lambda x : \tau.t_b)\uparrow^n_k : \sigma$ by induction hypothesis for $t_b\uparrow^n_{k+1}$ and by typing rule for abstraction.
- $t = @(t_l, t_r)$. We conclude $\Gamma\uparrow^n_k \vdash @(t_l, t_r)\uparrow^n_k : \sigma$ by induction hypothesis for $t_l\uparrow^n_k$, induction hypothesis for $t_r\uparrow^n_k$ and by typing rule for application. $\square$

The definitions and results for lowering are dual except that this time we need to take care of not lowering indices below 0.

Definition 8.27. [Term lowering, $t\downarrow^n_k$] For a term t and $n, k \in \mathbb{N}$ we define its lowered version with variables with index less than k untouched and remaining ones decreased by n ($t\downarrow^n_k$) as:

$$\begin{aligned}
f\downarrow^n_k &= f \\
x\downarrow^n_k &= x && \text{if } x < k \\
x\downarrow^n_k &= x \mathbin{\dot-} n && \text{if } x \geqslant k \\
@(t_l, t_r)\downarrow^n_k &= @(t_l\downarrow^n_k, t_r\downarrow^n_k) \\
\lambda x : \sigma.t\downarrow^n_k &= \lambda x : \sigma.t\downarrow^n_{k+1}
\end{aligned}$$

We also define $t\downarrow^n := t\downarrow^n_0$. $\diamond$

Definition 8.28. [Environment lowering, $\Gamma\downarrow^n_k$] For an environment $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ we define its lowered version $\Gamma\downarrow^n_k$ as:

$$\Gamma\downarrow^n_k := \{x_i : \sigma_i \mid x_i : \sigma_i \in \Gamma, k > i \in \mathbb{N}\} \cdot \{(x_i \mathbin{\dot-} n) : \sigma_i \mid x_i : \sigma_i \in \Gamma, k + n \leqslant i \in \mathbb{N}\}$$

We also define $\Gamma\downarrow^n := \Gamma\downarrow^n_0$. $\diamond$

We have similar result to that of Lemma 8.26 just this time we need to make sure that in the initial term indices $k, \ldots, k+n-1$ are unused.

Lemma 8.29. *For any $n, k \in \mathbb{N}$ and any term $\Gamma \vdash t : \sigma$ if $\forall_{i \in \{k,\ldots,k+n-1\}}\ x_i \notin \mathit{Var}(\Gamma)$ then $\Gamma\!\downarrow^n_k \vdash t\!\downarrow^n_k : \sigma$.*

Proof. Structural induction on t. The proof is similar to the proof of Lemma 8.26. We use the assumption that $x_i \notin \mathit{Var}(\Gamma)$ for $k \leqslant i < k+n$ in the variable case to ensure that by lowering no variable declarations are lost. □

Coq ▸ The `Coq` formalization follows the structure of this presentation. So firstly the definition operating on un-typed terms (`prelift` and `prelower`) are introduced. Then operations computing lifted (resp. lowered) version of the environment (`liftedEnv` and `loweredEnv`). Finally it is proven that lifted (resp. lowered) pseudoterms are well-typed in the respective environment. ◂

8.3.3 Definition of Substitution

Now we can present the definition of substitution. First we present it for un-typed terms.

Definition 8.30. [Substitution] A *substitution* is a finite set of pairs of variables and typed terms:

$$\tau = \{x_1/\Gamma_1 \vdash t_1 : \sigma_1, \ldots, x_n/\Gamma_n \vdash t_n : \sigma_n\}$$

such that for all $i \neq j \in \{1, \ldots, n\}$, $x_i \neq x_j$.

A *substitution domain* is defined as an environment: $\mathit{Dom}(\tau) = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ and, in case all environments Γ_i for $i \in \{1, \ldots, n\}$ are pairwise compatible[3], we also define a *substitution range* as an environment: $\mathit{Ran}(\tau) = \bigcup_{i \in \{1,\ldots,n\}} \Gamma_i$. Abusing notation we will also write $x \in \mathit{Dom}(\Gamma)$ for $x \in \mathit{Var}(\mathit{Dom}(\Gamma))$.

By $\tau_{\setminus \mathcal{X}}$ we denote the substitution τ with its domain restricted to $\mathcal{V} \setminus \mathcal{X}$, that is:

$$\tau_{\setminus \mathcal{X}} = \{(x_i/\Gamma_i \vdash t_i : \sigma_i) \in \tau \mid i \in \{1, \ldots, n\}, x_i \notin \mathcal{X}\} \qquad \diamond$$

Definition 8.31. [Substitution on un-typed terms] We define *substitution on un-typed terms* as follows:

$$\begin{aligned}
x\tau &= x && \text{if } x \notin \mathit{Dom}(\tau)\\
x\tau &= u && \text{if } x/\Gamma \vdash u : \sigma \in \tau\\
f\tau &= f &&\\
@(t_l, t_r)\tau &= @(t_l\tau, t_r\tau) &&\\
(\lambda x\!:\!\sigma.t)\tau &= \lambda x\!:\!\sigma.t\tau_{\setminus\{x\}} &&
\end{aligned} \qquad \diamond$$

[3] Which will be the case for compatible substitutions, see Definition 8.33.

The computation of $\tau_{\setminus\{x\}}$ in de Bruijn notation is realized via taking $\tau\uparrow^1$: the lifted version of substitution τ, with lifting operation on substitution defined as follows:

Definition 8.32. [Substitution lifting] Let $\tau = \{x_1/\Gamma_1 \vdash t_1 : \sigma_1, \ldots, x_n/\Gamma_n \vdash t_n : \sigma_n\}$ be a substitution. We define its lifted version as:

$$\tau\uparrow^n = \{(x_1 + n)/(\Gamma_1 \vdash t_1 : \sigma_1)\uparrow^n, \ldots, (x_n + n)/(\Gamma_n \vdash t_n : \sigma_n)\uparrow^n\} \qquad \diamond$$

Substitution operates on typed terms and hence is not always applicable as there may be type and environment clashes. The following definition captures conditions that are required for a substitution to be applicable to a term.

Definition 8.33. [Compatibility of substitution] We say that a substitution $\tau = \{x_1/\Gamma_1 \vdash t_1 : \sigma_1, \ldots, x_n/\Gamma_n \vdash t_n : \sigma_n\}$ is *compatible* with a term $\Gamma \vdash t : \sigma$ if the following conditions are satisfied:

- Environments of terms in τ are compatible:
  $\forall_{i \neq j \in \{1,\ldots,n\}}\ \Gamma_i \rightleftharpoons \Gamma_j$.
- Domain of τ is compatible with the environment of t:
  $\Gamma \rightleftharpoons \mathit{Dom}(\tau)$.
- Declarations in the range of τ not present in the domain of τ are compatible with the environment of t:
  $\Gamma \rightleftharpoons \mathit{Ran}(\tau) \setminus \mathit{Dom}(\tau)$. $\diamond$

The following result ensures that the conditions posted in the above definition are sufficient to type the result of application of substitution. This is a stronger version of the result from [JR01]. We need two auxiliary lemmas first.

Lemma 8.34. *Let $\tau = \{x_1/\Gamma_1 \vdash t_1 : \sigma_1, \ldots, x_n/\Gamma_n \vdash t_n : \sigma_n\}$ be a substitution such that all terms in τ have compatible environments, that is: $\forall_{i \neq j \in \{1,\ldots,n\}}\ \Gamma_i \rightleftharpoons \Gamma_j$. Then for any i, $\mathit{Ran}(\tau) = \mathit{Ran}(\tau) \cdot \Gamma_i$.*

Proof. The result follows from the fact that environment composition is idempotent and commutative for compatible environments. □

Lemma 8.35. *Let $\tau = \{x_1/\Gamma_1 \vdash t_1 : \sigma_1, \ldots, x_n/\Gamma_n \vdash t_n : \sigma_n\}$ be a substitution such that all terms in τ have compatible environments, that is: $\forall_{i \neq j \in \{1,\ldots,n\}}\ \Gamma_i \rightleftharpoons \Gamma_j$. Then for any i, $\mathit{Ran}(\tau) \vdash t_i : \sigma_i$.*

Proof. By Lemma 8.34, $\mathit{Ran}(\tau) = \mathit{Ran}(\tau) \cdot \Gamma_i$ and then $\mathit{Ran}(\tau) \cdot \Gamma_i \vdash t_i : \sigma_i$ by Lemma 8.13. □

Theorem 8.36. *Let $\Gamma \vdash t : \sigma$ be a term and let $\tau = \{x_1/\Gamma_1 \vdash t_1 : \sigma_1, \ldots, x_n/\Gamma_n \vdash t_n : \sigma_n\}$ be a substitution compatible with this term. Then:*

$$(\Gamma \setminus \mathit{Dom}(\tau)) \cdot \mathit{Ran}(\tau) \vdash t\tau : \sigma$$

Proof. Structural induction on t.

- $t = x$.
  - If $x \in Dom(\tau)$ then $x/\Gamma_i \vdash t_i : \sigma_i \in \tau$ for some i and $x\tau = t_i$. By Lemma 8.35 we get that $Ran(\tau) \vdash t_i : \sigma_i$, by Lemma 8.13 $(\Gamma \setminus Dom(\tau)) \cdot Ran(\tau) \vdash t_i : \sigma_i$ and finally $\sigma_i = \sigma$ by the assumption on compatibility of τ with t (hence domain of τ is compatible with t).
  - If $x \notin Dom(\tau)$ then $x\tau = x$ and to conclude the result by the typing rule for variable we need to show that $(\Gamma \setminus Dom(\tau)) \cdot Ran(\tau) \vdash x : \sigma$. Either $x : \tau \in Ran(\Gamma)$ but then $\tau = \sigma$ by compatibility of Γ with $Ran(\tau) \setminus Dom(\tau)$ (τ is compatible with t; note that $x \notin Dom(\tau)$). Or $x \notin Var(Ran(\Gamma))$ but then $x : \sigma \in \Gamma$ and hence $x : \sigma \in (\Gamma \setminus Dom(\tau)) \cdot Ran(\tau)$.
- $t = f$. Then $f\tau = f$ and $(\Gamma \setminus Dom(\tau)) \cdot Ran(\tau) \vdash f : \sigma$ by the typing rule for constant.
- $t = @(t_l, t_r)$ with $\Gamma \vdash t_l : \xi \to \sigma$ and $\Gamma \vdash t_r : \xi$. Then $(\Gamma \setminus Dom(\tau)) \cdot Ran(\tau) \vdash t_l\tau : \xi \to \sigma$ and $(\Gamma \setminus Dom(\tau)) \cdot Ran(\tau) \vdash t_r\tau : \xi$ by the induction hypothesis and $(\Gamma \setminus Dom(\tau)) \cdot Ran(\tau) \vdash @(t_l, t_r)\tau : \sigma$ by the typing rule for application as $@(t_l, t_r)\tau = @(t_l\tau, t_r\tau)$.
- $t = \lambda x{:}\tau.t_b$. Substitution τ is compatible with $\Gamma \vdash t : \sigma$ so $\Gamma \rightleftharpoons Dom(\tau)$ and hence $\Gamma \cdot \{x : \tau\} \rightleftharpoons Dom(\tau_{\setminus\{x\}})$. Similarly $\Gamma \rightleftharpoons Ran(\tau) \setminus Dom(\tau)$ and hence $\Gamma \cdot \{x{:}\tau\} \rightleftharpoons Ran(\tau_{\setminus\{x\}}) \setminus Dom(\tau_{\setminus\{x\}})$. So substitution $\tau_{\setminus\{x\}}$ is compatible with the term $\Gamma \cdot \{x : \tau\} \vdash t_b : \tau \to \sigma$ and by the induction hypothesis we get $(\Gamma \cdot \{x{:}\tau\} \setminus Dom(\tau_{\setminus\{x\}})) \cdot Ran(\tau_{\setminus\{x\}}) \vdash t_b\tau_{\setminus\{x\}} : \tau \to \sigma$. By the typing rule for abstraction and by observation that $(\Gamma \setminus Dom(\tau)) \cdot Ran(\tau) \cdot \{x{:}\tau\} = (\Gamma \cdot \{x{:}\tau\} \setminus Dom(\tau_{\setminus\{x\}})) \cdot Ran(\tau_{\setminus\{x\}})$ we conclude $(\Gamma \setminus Dom(\tau)) \cdot Ran(\tau) \vdash \lambda x{:}\tau.t_b\tau : \sigma$. □

Coq ▸ The part concerning substitution is by far the largest part of the development of $\lambda^{\to}$. That is primarily because indeed the definition of substitution on typed terms is rather complex. But also the development contains some more results about substitution not included in this presentation.

Because of the use of de Bruijn indices substitution is simply a list of `option Term` (`None` indicating that the given index is not in the domain of a given substitution). The actual substitution operating on typed terms corresponds to Theorem 8.36. ◂

8.4 Convertibility of Terms

For the development of computability in Section 9.2 we will need to define computability modulo classes of terms that are equivalent with respect to a relation on terms $\sim$, adhering to the following properties:

(i) it extends α-convertibility, so for α-convertible terms $t =_\alpha u$ we want to have $t \sim u$,

(ii) it relates terms that differ only on some additional declarations in environments that are not used, so we want to have $\Gamma \vdash t : \sigma \sim \Gamma' \vdash t : \sigma$ if $\Gamma \rightleftharpoons \Gamma'$,

(iii) finally we want to relate terms that differ only on names of free variables that is we want to have $t \sim u$ if there exist a renaming of variables τ such that $t = u\tau$. Note that in de Bruijn notation such renaming corresponds to permutation of indexes of free variables.

If $t \sim u$ then we will say that t and u are $\sim$-convertible.

We shortly present motivation for those three requirements:

(i) Typically we do not want to distinguish α-convertible terms in any way. This is also the easiest requirement as we are using de Bruijn indices to represent terms and in this representation α-convertible terms are simply equal.

(ii) The typical reasoning in computability proofs will be as follows: "given term $\Gamma \vdash t : \sigma \to \tau$ take variable $\Gamma \cdot \{x : \sigma\} \vdash x : \sigma$ and consider application $\Gamma \cdot \{x : \sigma\} \vdash @(t, x) : \tau \ldots$". Note that constructing such application requires extending Γ with the declaration for x. On the other hand we would like to have that the left argument of this application is equal to t. Strictly speaking it is not equal as it has an extended environment. But thanks to (ii) it will be $\sim$-convertible.

(iii) The reason behind this requirement is to have $\sim$-convertibility of lifted terms, so: $t \sim t\!\uparrow^i$ for any i. This in turn is needed for substitution. We will assume in Section 9.2 to have a substitution with all terms in its domain computable. But those terms are being substituted in context of some abstractions and hence need to be lifted (as explained in Section 8.3.2). So we want to be able to conclude computability of those lifted terms and since we are defining convertibility relation anyhow, solving this problem by demanding that terms and their liftings are convertible seems to be rather natural.

We will spend the rest of this section seeking convertibility relation on terms $\sim$ satisfying posed requirements. The idea is, roughly speaking, to define two terms to be convertible if there exist an endomorphism on free variables of one term that maps them to free variables of the other term. We begin with the definition of such variable mappings.

Definition 8.37. [Variable mapping] We say that a partial function $\Phi : \mathcal{V} \to \mathcal{V}$ is a *variable mapping* if it is injective.

Since Φ is an injective function there exist its inverse Φ^{-1} and since this symmetry will play an important role we will write variable mappings using infix notation so $x \ \Phi \ y$ instead of $\Phi(x) = y$. $\diamond$

Now given such variable mapping we can say when two environments or two un-typed terms are convertible modulo this mapping.

Definition 8.38. [Environment convertibility] Γ and Δ are *convertible environments* modulo variable mapping Φ, denoted as: $\Gamma \overset{\Phi}{\approx} \Delta$, if:

$$\forall_{x:\sigma\in\Gamma,\ y:\tau\in\Delta}\ x\ \Phi\ y \implies \sigma = \tau \qquad \diamond$$

Definition 8.39. [Un-typed terms convertibility] We define t and t' to be *convertible un-typed terms* modulo variable mapping Φ, denoted as $t \overset{\Phi}{\approx} t'$[4], inductively as:

$$\begin{array}{rcl} x \overset{\Phi}{\approx} y & & \text{if } x\ \Phi\ y \\ f \overset{\Phi}{\approx} f & & \\ @(t_l, t_r) \overset{\Phi}{\approx} @(u_l, u_r) & & \text{if } t_l \overset{\Phi}{\approx} u_l \text{ and } t_r \overset{\Phi}{\approx} u_r \\ \lambda x\!:\!\sigma.t \overset{\Phi}{\approx} \lambda x\!:\!\sigma.u & & \text{if } t \overset{\Phi\uparrow^1}{\approx} u \end{array} \qquad \diamond$$

Now it seems that we can say that two terms $\Gamma \vdash t : \sigma$, $\Delta \vdash u : \tau$ are convertible ($t \overset{\Phi}{\sim} u$) if there exists a variable mapping Φ such that $\Gamma \overset{\Phi}{\approx} \Delta$ and $t \overset{\Phi}{\approx} u$. However we need to be careful. If we require convertibility of full environments then the following desired property does not hold:

$$t \overset{\Phi}{\sim} u \ \wedge\ \Phi \subset \Phi' \implies t \overset{\Phi'}{\sim} u$$

To see that consider terms: $t = x : \sigma \vdash c : \delta$ and $u = x : \tau \vdash c : \delta$ and notice that we have $t \sim_{\varnothing} u$ but $t \not\sim_{\{(x,x)\}} u$ as environments of t and u declare x with different types.

This can be easily repaired if we demand convertibility of environments on free variables only, that is only those declarations that are really used in given term. The definition of term convertibility follows:

Definition 8.40. [Term convertibility, $\sim$] Terms $\Gamma \vdash t : \sigma$ and $\Gamma' \vdash t' : \sigma'$ are *convertible* up to variable mapping Φ, denoted as $\Gamma \vdash t : \sigma \overset{\Phi}{\sim} \Gamma' \vdash t' : \sigma'$ (we will often leave environments and types implicit and write $t \overset{\Phi}{\sim} t'$) iff:

$$\mathrm{Vars}(\Gamma \vdash t : \sigma) \overset{\Phi}{\approx} \mathrm{Vars}(\Gamma' \vdash t' : \sigma') \ \wedge\ t \overset{\Phi}{\approx} t'$$

[4] We abuse the notation here and denote un-typed term convertibility and environment convertibility with the same symbol $\approx$, however depending on the arguments being used it will always be clear which one is to be used.

Terms $\Gamma \vdash t : \sigma$ and $\Gamma' \vdash t' : \sigma'$ are *convertible* if there exist a variable mapping Φ such that $\Gamma \vdash t : \sigma \overset{\Phi}{\sim} \Gamma' \vdash t' : \sigma'$. $\diamond$

Then we extend the notion of convertibility to substitutions.

Definition 8.41. [Convertible substitutions, $\sim$] Substitutions τ and δ are *convertible* with variable mapping Φ, $\tau \overset{\Phi}{\sim} \delta$, iff:

$$\forall_{x,y \in \mathcal{V}}\ x\ \Phi\ y \implies \begin{cases} x \in Dom(\tau) \iff y \in Dom(\delta) \\ x/t \in \tau \wedge y/u \in \delta \implies t \overset{\Phi}{\sim} u \end{cases}$$

$\diamond$

Theorem 8.42. *The relation $\sim$ is an equivalence relation.*

Proof. For reflexivity, we have $t \overset{\Phi}{\sim} t$ with Φ being identity on variables restricted to free variables of t. For symmetry, if $t \overset{\Phi}{\sim} u$ then $u \overset{\Phi^{-1}}{\sim} t$. Finally transitivity is ensured as if $t \overset{\Phi}{\sim} u$ and $u \overset{\Psi}{\sim} w$ then $t \overset{\Phi \cdot \Psi}{\sim} w$. $\square$

The most important results concerning $\sim$ include:

- Compatibility with substitution: $t \sim t'$ and $\tau \sim \tau'$ implies $t\tau \sim t'\tau'$.
- Compatibility with beta-reduction, see Lemma 8.46.
- Compatibility with HORPO, see Lemma 9.20.

Coq ▸ The most interesting aspect of this part of the development is probably the representation of variable mappings in Coq. Variable mappings are partial, injective functions. Moreover we need to be able to compute their inverse for proving symmetry of $\sim$. We know that the inverse of any variable mappings exists, as it is an injective function. But this does not make our task any easier as we want to provide a constructive proof and for that we need to be able to compute this inverse: something that clearly cannot be done in full generality. But before giving up constructiveness let us observe that variable mappings operate on environments which are finite. So both domain and codomain of variable mappings are finite and computing inverse of such functions can be accomplished.

To encode variable mappings in Coq we have chosen to model Φ as a relation. Then computing inverse is trivial as we only need to transpose the relation but we still need to make sure that we can compute $\Phi(x)$ for any x. Let us first present the solution that we employed and then we will discuss it.

```
Record EnvSubst : Type := build_envSub {
 envSub:     relation nat;
```

```
 size:       nat;
 envSub_dec: forall i j, {envSub i j} + {~envSub i j};
 envSub_Lok: forall i j j', envSub i j -> envSub i j' -> j = j';
 envSub_Rok: forall i i' j, envSub i j -> envSub i' j -> i = i';
 sizeOk:     forall i j, envSub i j -> i < size /\ j < size
}.
```

So `envSub` represents Φ function (seen as a relation). Fields `envSub_Lok` and `envSub_Rok` ensure that `envSub` is, respectively, a function and that it is injective. The `size` field is an upper bound on indices of variables both in the domain and the codomain of `envSub` and `sizeOk` verifies that indeed that is the case. Finally `envSub_dec` states that `envSub` relation is decidable.

Now to compute $\Phi(x)$ we check whether `envSub x y` holds (with the use of `envSub_dec`) for $y \in \{0, \ldots, size - 1\}$. If we find such y then $\Phi(x) = y$ and we know that this y is unique by `envSub_Lok`. On the other hand if no such y exists in this interval then we know that it does not exist at all due to `sizeOk` and we conclude that x is not in the domain of Φ. So using this reasoning we can prove the following lemma:

```
Lemma envSubst_dec: forall (i: nat) (Q: EnvSubst),
  {j: nat | envSub Q i j} + {forall j, ~envSub Q i j}.
```

Using such representation of variable mappings the remaining definitions of this section can be naturally expressed in Coq. However, working on structures for which Leibniz equality does not denote the intended equality is not very easy in Coq. Setoid is an extension that makes it somehow easier by allowing to register an equivalence relation along with some functions compatible with it (morphisms). Then one can replace a term by an equivalent one in arguments of such functions as easily as if they were equal. The convertibility relation $\sim$ was proven to be an equivalence relation (actual Coq proofs are somehow more complicated than what the proof of Theorem 8.42 would suggest) and registered as a setoid. Then we proved a number of operations to be morphisms with respect to $\sim$. ◂

8.5 β-reduction

The β-reduction relation expresses the way in which the application of a function to arguments is evaluated and hence is a model of computation for $\lambda^{\rightarrow}$.

Definition 8.43. $[\rightarrow_\beta]$ The *β-reduction rule* is defined as:

$$@(\lambda x{:}\sigma.t, u) \rightarrow_\beta t[x/u]$$

The *β-reduction relation* is the smallest relation on terms that satisfies β-reduction rule and is closed under contexts:

$$\frac{t \to_\beta t'}{\lambda x\!:\!\sigma.t \to_\beta \lambda x\!:\!\sigma.t'} \qquad \frac{t \to_\beta t'}{@(t,u) \to_\beta @(t',u)} \qquad \frac{u \to_\beta u'}{@(t,u) \to_\beta @(t,u')} \qquad \diamond$$

Theorem 8.44 (Subject reduction)**.** *If* $\Gamma \vdash t : \sigma \to_\beta \Gamma \vdash u : \tau$ *then* $\sigma = \tau$. □

Lemma 8.45. *β-reduction preserves variables, that is:*

$$t \to_\beta u \implies \mathrm{Vars}(t) \supseteq \mathrm{Vars}(u)$$ □

Lemma 8.46. *β-reduction is compatible with* $\sim$*, that is:*

$$\left.\begin{array}{c} \Gamma \vdash t : \delta \to_\beta \Delta \vdash u : \eta \\ \Gamma \vdash t : \delta \overset{Q}{\sim} \Gamma' \vdash t' : \delta' \\ \Delta \vdash u : \eta \overset{Q}{\sim} \Delta' \vdash u' : \eta' \\ \Gamma' = \Delta' \end{array}\right\} \implies \Gamma' \vdash t' : \delta' \to_\beta \Delta' \vdash u' : \eta'$$

Proof. Induction on t. All cases but β-reduction step at the root easily by the induction hypothesis. For β-reduction step at the root $@(\lambda x : \tau.s, w) \to_\beta s[x/w]$ we get $t' = @(\lambda x\!:\!\tau.s', w')$ and $u' = s[x/w']$ with $s \overset{Q\uparrow^1}{\sim} s'$ and $w \overset{Q}{\sim} w'$ and hence $t' \to_\beta u'$. □

Lemma 8.47. *β-reduction is stable under substitution, that is:*

$$t \to_\beta u \implies t\tau \to_\beta u\tau$$ □

We will need the following simple lemma in Section 9.3.

Lemma 8.48.

$$f(t_1, \ldots, t_n) \to_\beta u \implies \exists_{i \in \{1,\ldots,n\}}\ u = f(t_1, \ldots, t_i', \ldots, t_n) \wedge t_i \to_\beta t_i'$$

Proof. Follows easily from

$$@(t_1, \ldots, t_n) \to_\beta u \implies \exists_{i \in \{1,\ldots,n\}}\ u = @(t_1, \ldots, t_i', \ldots, t_n) \wedge t_i \to_\beta t_i'$$

that is easily proven by induction on n. □

Lemma 8.49. *β-reduction is monotonous, that is:*

$$u \to_\beta u' \implies t[u]_p \to_\beta t[u']_p$$ □

Coq ▸ The definition of β-reduction is done in two steps; first an arbitrary reduction compatible with term structure is defined. It is parameterized by a relation R. So a reduction is either a direct R-step at the root or a reduction in left or right argument of an application or a reduction in a body of an abstraction. Then such reduction is specialized to define the beta reduction relation. ◂

Chapter 9

Termination of Higher-Order Rewriting

In this chapter we define the higher-order recursive path ordering (HORPO) and prove some of its properties. We begin by introducing higher-order rewriting in Section 9.1. Then we present the computability predicate proof method due to Tait and Girard [GTL89, Tai67] in Section 9.2. Finally we define HORPO in Section 9.3 and prove some of its properties, most notably well-foundedness of the union of HORPO and the beta-reduction of the $\lambda^{\rightarrow}$, which requires use of the aforementioned computability method.

9.1 Higher-Order Rewriting

In this section we introduce the concept of higher-order rewriting, that is rewriting terms with bound variables. In Section 9.1.1 we introduce algebraic terms that we will use in Section 9.1.2 where we present the AFS format for higher-order rewriting and shortly discuss its relation with other existing formats.

9.1.1 Terms

In this section we will introduce algebraic terms. The main difference with λ-terms as introduced in Section 8.1 is that now function symbols are algebraic operators

An earlier version of this chapter appeared as: A. Koprowski, Certified Higher-Order Recursive Path Ordering, In F. Pfenning ed., *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, Seattle, WA, USA, volume 4098 of Lecture Notes in Computer Science, pp. 227–241, Springer-Verlag, August 2006.

equipped with arity. We use the definition of simple types $\mathcal{T}_{\mathcal{S}}$ over a given set of sorts $\mathcal{S}$ from Section 8.1. Let us begin with the definition of algebraic signature.

Definition 9.1. [Algebraic signature] A declaration of a function symbol f expecting n arguments of types $\alpha_1, \ldots, \alpha_n$ and an output type β we will write as $f : \alpha_1 \times \ldots \times \alpha_n \to \beta$.

An *algebraic signature* $\mathcal{F}$ is a set of such function declarations. $\diamond$

Now we define algebraic terms as follows:

Definition 9.2. [Algebraic terms] A set of *algebraic terms* is defined by the following grammar:

$$\mathcal{P}t_a := \mathcal{V} \mid @(\mathcal{P}t_a, \mathcal{P}t_a) \mid \lambda\mathcal{V}{:}\mathcal{T}_{\mathcal{S}}.\mathcal{P}t_a \mid \mathcal{F}(\mathcal{P}t_a, \ldots, \mathcal{P}t_a)$$

and such terms conform to the following typing rules (compare with Definition 8.5):

$$\frac{x{:}\alpha \in \Gamma}{\Gamma \vdash x : \alpha} \qquad \frac{\begin{array}{c} f : \alpha_1 \times \ldots \times \alpha_n \to \beta \in \Sigma \\ \Gamma \vdash t_1 : \alpha_1, \ldots, \Gamma \vdash t_n : \alpha_n \end{array}}{\Gamma \vdash f(t_1, \ldots, t_n) : \beta}$$

$$\frac{\Gamma \vdash t : \alpha \to \beta \qquad \Gamma \vdash u : \alpha}{\Gamma \vdash @(t, u) : \beta} \qquad \frac{\Gamma \cup \{x{:}\alpha\} \vdash t : \beta}{\Gamma \vdash \lambda x{:}\alpha.t : \alpha \to \beta} \quad \diamond$$

Example 9.3. *Consider two sorts:* $\mathbb{N}$ *for natural numbers and* List *representing lists of natural numbers.*

$$\mathcal{S} = \{\mathbb{N}, \mathrm{List}\}$$

Now consider the following signature $\mathcal{F}$*:*

$$\begin{aligned} \mathcal{F} = \{ & \,\mathrm{nil} : \mathrm{List}, \\ & \mathrm{cons} : \mathbb{N} \times \mathrm{List} \to \mathrm{List}, \\ & \mathrm{map} : \mathrm{List} \times (\mathbb{N} \to \mathbb{N}) \to \mathrm{List}\} \end{aligned}$$

Some terms over this signature:

$$\begin{aligned} \varnothing &\vdash \mathrm{nil} : \mathrm{List} \\ X{:}\mathbb{N} \to \mathbb{N} &\vdash \mathrm{map}(\mathrm{nil}, X) : \mathrm{List} \\ x{:}\mathbb{N}, l{:}\mathrm{List}, X{:}\mathbb{N} \to \mathbb{N} &\vdash \mathrm{map}(\mathrm{cons}(x, l), X) : \mathrm{List} \\ x{:}\mathbb{N}, l{:}\mathrm{List}, X{:}\mathbb{N} \to \mathbb{N} &\vdash \mathrm{cons}(@(X, x), \mathrm{map}(l, X)) : \mathrm{List} \end{aligned} \quad \lhd$$

Note that in the formalization our intention is to use $\lambda^{\to}$ terms to represent such algebraic terms. To avoid dealing with arities we made a simplification and assumed that output types of functions are base types, an assumption often made in the literature. This allows us to represent a function $f : \alpha_1 \times \ldots \times \alpha_n \to \beta$ by a $\lambda^\to$ constant $f : \alpha_1 \to \ldots \to \alpha_n \to \beta$ and its application $f(t_1, \ldots, t_n)$ as a $\lambda^\to$ term $@(f, t_1, \ldots, t_n)$. This also means that we restrict to fully applied terms, as in Definition 8.7.

Remark. All the theory in the following sections deals with algebraic terms. They are assumed to be encoded in this way and hence all the definitions of substitution, positions etc. from Chapter 8 need not be repeated here for this new algebraic structure.

Coq ▸ As remarked above the higher-order terms with arity were encoded in `Coq` using simply typed lambda-terms. So effectively algebraic terms are a subset of simply typed lambda terms where every function is fully applied. Please note that this puts no restriction on types and/or level of application for variables. This condition has been formalized by the following predicate:

```
Inductive algebraic: Term -> Prop :=
| AlgVar: forall M, isVar M -> algebraic M
| AlgAbs: forall M (Mabs: isAbs M), algebraic (absBody Mabs) -> algebraic M
| AlgApp: forall M, isApp M -> ~isFunApp M ->
     (forall M', isAppUnit M' M -> algebraic M') -> algebraic M
| AlgFunApp: forall M, isFunApp M -> isBaseType (type M) ->
     (forall M', isArg M' M -> algebraic M') -> algebraic M.
```

Now every time an algebraic term is expected we take a pair of a lambda term (`T: Term`) and a proof that it satisfies this condition (`Talg: algebraic T`). Another possibility would be to introduce a type for algebraic terms as a refinement of lambda terms and then introduce a coercion between the two.

```
Definition ATerm := { T: Term | algebraic T }.
Definition aterm2term (A: ATerm) : Term := proj1_sig A.
Coercion aterm2term : ATerm >-> Term.
```

◂

9.1.2 Rewriting

There are several variants of higher-order rewriting. Here we use the algebraic-functional systems (AFSs) introduced by Jouannaud and Okada [JO91]. The main difference between AFSs and another popular format of higher-order rewriting systems (HRSs, [Nip91]) is that in HRSs we work modulo beta-eta (using pure $\lambda^\to$ terms) whereas in AFSs we do not (and function symbols have fixed arity, as in Definition 9.2). As a consequence rewriting for AFSs is defined using plain pattern matching compared to rewriting modulo $\beta\eta$ of $\lambda^\to$ in HRSs framework. For a broader discussion on this subject we refer the reader to, for instance, [Raa03]. The presentation in this section follows [JR01].

We give definitions of higher-order: rewrite rules, rewriting systems and rewrite relation.

Definition 9.4. [Higher-order term rewriting system] Given a signature $\mathcal{F}$ a *rewrite rule* is a quadruple $\Gamma \vdash \ell \to r : \alpha$ where ℓ and r are algebraic terms such that:

- $\mathrm{Vars}(r) \subseteq \mathrm{Vars}(l)$
- $\Gamma \vdash \ell : \alpha$ and $\Gamma \vdash r : \alpha$.

A *higher-order rewriting system* is a set of rewrite rules. $\diamond$

Definition 9.5. [The rewrite relation] Given higher-order rewriting system R a term $\Gamma \vdash s : \alpha$ rewrites to a term $\Gamma \vdash t : \alpha$ if there exist a rewrite rule $\Delta \vdash \ell \to r : \beta \in R$, a substitution γ and a position p such that:

- $\mathit{Dom}(\gamma) \subseteq \Delta$,
- $\Delta \cdot \mathit{Ran}(\gamma) \subseteq \Gamma_{s_{|p}}$,
- $s_{|p} = \ell\gamma$,
- $t = s[r\gamma]_p$. $\diamond$

Example 9.6. *[Example 9.3 continued] With signature given in Example 9.3 we can construct the following higher-order term rewriting system (from [JR99]) representing the usual map computation on lists of natural numbers:*

$$\begin{array}{rcl} X : \mathbb{N} \to \mathbb{N} \vdash & \mathrm{map}(\mathrm{nil}, X) \to \mathrm{nil} & : \ \mathrm{List} \\ x : \mathbb{N}, l : \mathrm{List}, X : \mathbb{N} \to \mathbb{N} \vdash & \mathrm{map}(\mathrm{cons}(x, l), X) \to \mathrm{cons}(@(X, x), \mathrm{map}(l, X)) & : \ \mathrm{List} \end{array}$$

$\lhd$

For more detailed introduction to higher-order rewriting in AFS format and proving its termination by means of higher-order reduction orderings we refer the reader to [JR01].

Coq ▸ Note that Definitions 9.4 and 9.5 do not play any direct role for our results and hence are omitted in the formalization. ◂

9.2 Computability

In this section we present the computability predicate proof method due to Tait and Girard [GTL89, Tai67]. In Section 9.3.3 we will use computability with respect to a particular relation (being the union of HORPO and β-reduction relation in that case) but here we present computability for an arbitrary relation satisfying given properties.

We begin by giving a definition of computability in Section 9.2.1 and in Section 9.2.2 we prove some computability properties that we will need in Section 9.3.

9.2.1 Definition of Computability

We begin by presenting the definition of computability predicate.

Definition 9.7. [Computability] A term $\Gamma \vdash t : \delta$ is computable with respect to a relation on terms $\gg$, denoted as $t \in \mathbb{C}_\delta$ (or simply $t \in \mathbb{C}$ if the type is of no interest), if:

- δ is a base type and t is strongly normalizable with respect to $\gg$ ($t \in \mathcal{A}cc_{\gg}$), or
- $\delta = \sigma \to \tau$ and $@(t', u) \in \mathbb{C}_\tau$ for all $t' \sim t$ and all $u \in \mathbb{C}_\sigma$. $\diamond$

This definition deserves a few words of explanation. Firstly, it is usual to assume that variables are computable. We do not do that, following the presentation in [JR99] and we prove that variables are computable as one of the computability properties.

Another deviation from the standard definition is the fact that we define computability modulo convertibility relation on terms ($\sim$). That is because a typical pattern in computability proof will be as follows: "for $t \in \mathbb{C}_{\sigma\to\tau}$ take a fresh variable $x : \sigma$ and consider $@(t, x) : \tau$ having $@(t, x) \in \mathbb{C}_\tau$ from the definition of computability, as variables are computable". But constructing the application $@(t, x)$ requires extending environment of t with a declaration for x. Such subtleties are usually omitted in a presentation but our goal is to make presentation that closely reflects the formal verification that has been made. That is why we define computability modulo $\sim$, which will also prove helpful for dealing with computability of lifted terms as we shall see later.

Coq ▸ To begin with the notion of well-foundedness corresponding to Definition 1.7 is present in the standard library of Coq in the module `Coq.Init.Wf`. The membership in $\mathcal{W}_<^{Acc}$ is expressed by the accessibility predicate `Acc` and the induction principle generated by Coq corresponds to the one from Definition 1.8.

The coding of the definition of computability in Coq poses some technical difficulties. The problem is that it needs to be expressed as a fixpoint definition and Coq uses a simple criterion to ensure that such definitions are terminating, namely one of the arguments in the recursive call needs to be a subterm of the original argument. This is not the case for computability. To check whether $t : \sigma \to \tau$ is computable we check whether its application to a computable term $u : \sigma$ is computable. Although types of $u : \sigma$ and $@(t, u) : \tau$ are simpler than of $t : \sigma \to \tau$ this is not enough for a simple syntactic criterion of Coq. What makes matters even worse is that actually we do not take $@(t, u) : \tau$ but $@(t', u) : \tau$ with $t \sim t'$. Hence we extracted the type of a term as an extra argument to an auxiliary `ComputableS` function. This argument satisfies Coq requirements of decreasing arguments and hence Coq accepts this definition. Then `Computable` merely calls `ComputableS` with the appropriate type.

```
Fixpoint ComputableS (M: Term) (T: SimpleType)
  {struct T} : Prop := algebraic M /\ type M = T /\
  match T with
  | #T => AccR M
  | TL --> TR =>
    forall P (Papp: isApp P) (PL: appBodyL Papp ~ M)
      (typeL: type P = TR) (typeR: type (appBodyR Papp) = TL),
      algebraic (appBodyR Papp) ->
      ComputableS (appBodyR Papp) TL ->
      ComputableS P TR
  end.

Definition Computable M := ComputableS M (type M).
```

It is worth noting that a new `Function` feature of Coq 8.1, allowing for more complex fixpoint definitions where the obligation of proving that some argument is decreasing is left to the user, could be very helpful in this and many other similar situations. However at the time when this work was carried out this feature was not powerful enough to deal with our variant of computability. ◂

9.2.2 Computability Properties

We want to abstract away from the particular relation with respect to which we define computability. So let us assume an arbitrary relation on terms $\gg$ and in this section by $\mathcal{A}cc$ we will mean $\mathcal{A}cc_{\gg}$. But in order to prove required computability properties we need to make some assumptions about $\gg$. Table 9.1 presents the list of properties we require $\gg$ to conform to. All those properties but (P_8) are quite general and natural so our abstraction is (partly) successful. The property (P_8) looks rather complicated but basically it states that every reduction of an application either operates on separate arguments or it is a β-reduction step. This property is rather specific for a particular $\gg$ relation being in that case union of β-reduction and HORPO as we will use it in Section 9.3.

(P_1)	Subject reduction $t : \sigma \gg u : \tau \implies \sigma = \tau$
(P_2)	Preservation of environments $\Gamma_t \vdash t : \delta \gg \Gamma_u \vdash u : \eta \implies \Gamma_t = \Gamma_u$
(P_3)	Preservation of variables $t \gg u \implies \mathrm{Vars}(t) \supseteq \mathrm{Vars}(u)$
(P_4)	Normal form of variables $\neg(x \gg u)$
(P_5)	Compatibility with $\sim$ $\left.\begin{array}{c} \Gamma \vdash t : \delta \gg \Delta \vdash u : \eta \\ \Gamma \vdash t : \delta \overset{Q}{\sim} \Gamma' \vdash t' : \delta' \\ \Delta \vdash u : \eta \overset{Q}{\sim} \Delta' \vdash u' : \eta' \\ \Gamma' = \Delta' \end{array}\right\} \implies \Gamma' \vdash t' : \delta' \gg \Delta' \vdash u' : \eta'$
(P_6)	Stability under substitution $t \gg u \implies t\gamma \gg u\gamma$
(P_7)	Monotonicity $u \gg u' \implies t[u]_p \gg t[u']_p$
(P_8)	Reductions of applications $t = @(t_l, t_r) \gg u \implies$ $\left\{\begin{array}{c} \exists_{t_b}\ t = @(\lambda x{:}\sigma.t_b, t_r) \wedge u = t_b[x/t_r] \\ \vee \\ \exists_{u_l, u_r}\ u = @(u_l, u_r) \wedge \left(\begin{array}{c} t_l = u_l \wedge t_r \gg u_r \\ \vee \\ t_l \gg u_l \wedge t_r = u_r \\ \vee \\ t_l \gg u_l \wedge t_r \gg u_r \end{array}\right) \end{array}\right.$
(P_9)	Reductions of abstraction $\lambda x{:}\sigma.t_b \gg u \implies \exists_{u_b}\ u = \lambda x{:}\sigma.u_b \wedge t_b \gg u_b$

Table 9.1 Abstract properties required for $\gg$.

(C_1)	Every reduct of a computable term is computable. $t \in \mathbb{C}_\delta \wedge t \gg u \implies u \in \mathbb{C}_\delta$ Lemma 9.10
(C_2)	Every computable term is strongly normalizable. $t \in \mathbb{C}_\delta \implies t \in \mathcal{A}cc$ Lemma 9.11
(C_3)	Neutral term is computable iff its every reduct is computable. t-neutral $\implies ((\forall_u\ t \gg u \implies u \in \mathbb{C}_\delta) \iff t \in \mathbb{C}_\delta)$ Lemma 9.11
(C_4)	Variables are computable. $\forall_{x:\delta}\ x \in \mathbb{C}_\delta$ Lemma 9.12
(C_5)	Computability of abstractions $\forall_{u \in \mathbb{C}_\sigma}\ t[x/u] \in \mathbb{C}_\tau \implies (\lambda x\!:\!\sigma.t) \in \mathbb{C}_{\sigma \to \tau}$ Lemma 9.13
(C_6)	Term convertible with computable term is computable. $t \in \mathbb{C}_\delta \wedge t \sim t' \implies t' \in \mathbb{C}_\delta$ Lemma 9.14

Table 9.2 Computability properties.

Table 9.2 on the other hand presents the list of all the computability properties that we will need in the following section. We proceed with presenting proofs for those properties. We begin with two simple auxiliary lemmas.

Lemma 9.8. *Let* $@(t, u) \in \mathcal{A}cc$ *then* $t \in \mathcal{A}cc$.

Proof. Easy using monotonicity (P_7). □

Lemma 9.9. *Let* $t \in \mathcal{A}cc$ *and* $t \sim u$ *then* $u \in \mathcal{A}cc$.

Proof. Easy using compatibility with $\sim$ (P_5). □

Now we will prove that reducts of computable terms are computable again.

Lemma 9.10. (C_1) *Let* $\Gamma \vdash t : \delta$*, such that* $t \in \mathbb{C}_\delta$*. The for all terms* $\Delta \vdash u : \sigma$*, such that* $t \gg u$ *we have* $u \in \mathbb{C}_\sigma$*.*

Proof. Induction on δ. For the base case $t \in \mathbb{C}_\delta$ and δ is a base type so $t \in \mathcal{A}cc$ by the definition of computability. Since $t \in \mathcal{A}cc$ and $t \gg u$, $u \in \mathcal{A}cc$. By subject reduction for $\gg$ (P_1), $\delta = \sigma$, so $u \in \mathbb{C}_\sigma$ by the definition of computability.

For the induction step let $\sigma = \rho \to \tau$. By the definition of computability $u \in \mathbb{C}_{\rho\to\tau}$ if for every $s \in \mathbb{C}_\rho$, $@(u, s) \in \mathbb{C}_\tau$. $@(t, s) \in \mathbb{C}_\tau$ by the definition of computability and $@(t, s) \gg @(u, s)$ by monotonicity (P_7). Finally we conclude $@(u, s) \in \mathbb{C}_\tau$ by the induction hypothesis. □

Let us recall that we did not assume variables to be computable. Variables of a base type are computable due to the definition of computability and the assumption that variables are not reducible (P_4). But variables of a functional type are computable by property (C_3), which forbids us to prove the computability properties (C_2) and (C_3) separately.

Lemma 9.11. *For all terms* $\Gamma \vdash t : \delta$, $\Delta \vdash u : \delta$ *we prove that:*

(C_2) $t \in \mathbb{C}_\delta \implies t \in \mathcal{A}cc$

(C_3) *if* t *is neutral then* $(\forall_{w\delta}\ t \gg w \implies w \in \mathbb{C}_\delta) \iff t \in \mathbb{C}_\delta$

Proof. Induction on type δ. Note that '$\Leftarrow$' part of (C_3) is (C_1) so below we only prove the '$\Rightarrow$' part of this property.

- δ is a base type.
  - (C_2) $t \in \mathbb{C}_\delta$ and δ is a base type so $t \in \mathcal{A}cc$ by the definition of computability.
  - (C_3) $t : \delta$ so to show $t \in \mathbb{C}_\delta$ we need to show $t \in \mathcal{A}cc$ but for every w such that $t \gg w$ we have $w \in \mathbb{C}_\delta$ by assumption. Hence $w \in \mathcal{A}cc$ by the definition of computability and $t \in \mathcal{A}cc$.
- $\delta = \sigma \to \tau$
  - (C_2) Take variable $x : \sigma$ which is computable by induction hypothesis (C_3) as variables are not reducible by (P_4). Now consider application $\Gamma \cup \{x : \sigma\} \vdash @(t, x) : \tau$ which is computable by the definition of computability (note that $\Gamma \cup \{x : \sigma\} \vdash t : \delta \sim \Gamma \vdash t : \delta$). So $\Gamma \cup \{x : \sigma\} \vdash @(t, x) : \tau \in \mathcal{A}cc$ by induction hypothesis (C_2). Then $\Gamma \cup \{x : \sigma\} \vdash t : \delta \in \mathcal{A}cc$ by Lemma 9.8 and $\Gamma \vdash t : \delta \in \mathcal{A}cc$ by Lemma 9.9.
    Remark. From now on we will work modulo $\sim$ without stating it explicitly which greatly improves the readability of the proofs. The reader interested in all the details is encouraged to consult the Coq scripts.

(C_3) By the definition of computability $t \in \mathbb{C}_{\sigma\to\tau}$ if for every $s \in \mathbb{C}_\sigma$, $@(t,s) \in \mathbb{C}_\tau$. By induction hypothesis for (C_2), $s \in \mathcal{A}cc$ so we continue by well-founded inner induction on s with respect to $\gg$.

$@(t,s) : \tau$ is neutral so we can apply induction hypothesis for (C_3) and we are left to show that all reducts of $@(t,s)$ are computable. We do case analysis using (P_8). Since t is neutral and hence is not an abstraction, we can exclude the β-reduction case and we are left with the following cases:

- $@(t,s) \gg @(t',s)$ with $t \gg t'$. Then t' is computable as so is every reduct of t and application of two computable terms is computable by the definition of computability.
- $@(t,s) \gg @(t,s')$ with $s \gg s'$. We observe that $s' \in \mathbb{C}$ by (C_1) and since $s \gg s'$ we apply the inner induction hypothesis to conclude $@(t,s') \in \mathbb{C}_\tau$.
- $@(t,s) \gg @(t',s')$ with $t \gg t'$ and $s \gg s'$. Every reduct of t is computable so $t' \in \mathbb{C}_{\sigma\to\tau}$. By ($C_1$) $s' \in \mathbb{C}_\sigma$. Again application of two computable terms is computable. □

An easy consequence of the above lemma is the fact that all variables are computable.

Lemma 9.12 (C_4)**.** *For every variable $x{:}\delta$, $x \in \mathbb{C}_\delta$.*

Proof. Variables are neutral so we apply (C_3) and since variables are in normal forms (P_4) we conclude $x \in \mathbb{C}_\delta$. □

The following property deals with computability of abstractions.

Lemma 9.13 (C_5)**.** *Consider an abstraction $(\lambda x : \sigma.t) : \sigma \to \tau$. If for every $u \in \mathbb{C}_\sigma$, $t[x/u] \in \mathbb{C}_\tau$ then $(\lambda x{:}\sigma.t) \in \mathbb{C}_{\sigma\to\tau}$.*

Proof. By the definition of computability $\lambda x{:}\sigma.t$ is computable if for every $s \in \mathbb{C}_\sigma$, $@(\lambda x : \sigma.t, s) \in \mathbb{C}_\tau$. Note that $t \in \mathbb{C}$ by assumption because $t = t[x/x]$ and $x \in \mathbb{C}$ by (C_4). So by (C_2) both $t \in \mathcal{A}cc$ and $s \in \mathcal{A}cc$ and we proceed by well-founded part induction on a pair of computable terms (t,s) with respect to ordering $\rhd = (\gg, \gg)_{lex}$. Now, since $@(\lambda x{:}\sigma.t, s)$ is neutral, by (C_3) we are left to show that all its reducts are computable. Let us continue by considering possible reducts of this application using (P_9). So we have $@(\lambda x{:}\sigma.t, s) \gg u$ and the following cases to consider:

- $u = t[x/s]$. $u \in \mathbb{C}$ by the assumption.
- $u = @(\lambda x : \sigma.t, s')$ with $s \gg s'$. $u \in \mathbb{C}$ by the induction hypothesis for $(t,s') \lhd (t,s)$.

- $u = @(w, s)$ with $\lambda x : \sigma.t \gg w$. By (P_8) we know that this reduction is in the abstraction body of $\lambda x : \sigma.t$ so in fact $w = \lambda x : \sigma.t'$ with $t \gg t'$. We conclude computability of u by the induction hypothesis for $(t', s) \lhd (t, s)$.
- $u = @(w, s')$ with $\lambda x : \sigma.t \gg w$ and $s \gg s'$. As in the above case, by (P_8) we observe that $w = \lambda x : \sigma.t'$ with $t \gg t'$ and we conclude computability of u by the induction hypothesis for $(t', s') \lhd (t, s)$. □

We conclude with the following simple property.

Lemma 9.14 (C_6)**.** *If $t \in \mathbb{C}_\delta$ and $t \sim t'$ then $t' \in \mathbb{C}_\delta$*

Proof. If δ is a simple type then we apply Lemma 9.9. If δ is an arrow type then we conclude $t' \in \mathbb{C}_\delta$ directly from the definition of computability for t. Note that here we make use of the fact that we defined computability modulo $\sim$. □

Coq ▸ Proving computability properties turned out to be the most difficult part of the whole development. In its first version ([Kop04]) those properties were assumed as axioms. Completing the pursuit of making the development axiom-free and proving all computability properties turned out to be a very laborious task after which the size of Coq script tripled.

Strictly speaking, in terms of script size, the part of the formalization dealing with computability accounts for only slightly more than 5%. However, as those properties are at the heart of proofs concerning HORPO, providing proofs for them triggered many other developments.

This difficulty can be partially explained by the real complexity of the computability predicate proof method. Other factors that contributed to making this task difficult include:

- the fact that algebraic terms were encoded using pure $\lambda^{\rightarrow}$ terms,
- the necessity of defining computability modulo $\sim$.

For the clarity of presentation those issues are left implicit in the computability proofs presented in this section but in Coq proofs they had to be taken care of. Another aspect not visible in this presentation is the use of de Bruijn indices [Bru72] to represent terms. ◂

9.3 HORPO: Higher-Order Recursive Path Ordering

This section is devoted to the core of the work presented in the second part of this thesis: the results concerning the higher-order recursive path ordering (HORPO).

We begin by presenting the definition of HORPO in Section 9.3.1, then some of its properties in Section 9.3.2 and its main property – well-foundedness — in Section 9.3.3.

9.3.1 Definition of HORPO

As indicated in the introduction to the second part of this thesis, Chapter 6, the subject of our formalization is a slight variant of HORPO as presented in [JR99]. We begin by first presenting the formalized variant of the definition and then we discuss the differences compared to the original definition of Jouannaud and Rubio.

Originally the ordering is defined for algebraic-functional systems [JO91] (AFSs), where all function symbols have a fixed arity. As explained in Section 9.1, we will use pure terms of the simply-typed lambda calculus, restricted to fully applied terms (see Definition 8.7).

Definition 9.15. [The higher-order recursive path ordering, $\succ^{\mathsf{HORPO}}$] Assume a well-founded ordering on the set of function symbols $\rhd$, called a precedence. We define HORPO, $\succ^{\mathsf{HORPO}}$, on terms as, $\Gamma \vdash t : \delta \succ^{\mathsf{HORPO}} \Gamma \vdash u : \delta$ iff one of the following holds:

(H_1) $t = f(t_1, \ldots, t_n)$, $\exists_{i \in \{1,\ldots,n\}}\ t_i \succeq^{\mathsf{HORPO}} u$

(H_2) $t = f(t_1, \ldots, t_n)$, $u = g(u_1, \ldots, u_k)$, $f \rhd g$, $t \succ\!\!\succ^{\mathsf{HORPO}} \{u_1, \ldots u_k\}$

(H_3) $t = f(t_1, \ldots, t_n)$, $u = f(u_1, \ldots, u_k)$, $\{\{t_1, \ldots t_n\}\} \succ^{\mathsf{HORPO}}_{MUL} \{\{u_1, \ldots, u_k\}\}$

(H_4) $@(u_1, \ldots, u_k)$ is a partial left-flattening of u, $t \succ\!\!\succ^{\mathsf{HORPO}} \{u_1, \ldots u_k\}$

(H_5) $t = @(t_l, t_r)$, $u = @(u_l, u_r)$, $\{\{t_l, t_r\}\} \succ^{\mathsf{HORPO}}_{MUL} \{\{u_l, u_r\}\}$

(H_6) $t = \lambda x\!:\!\sigma.t'$, $u = \lambda x\!:\!\sigma.u'$, $t' \succ^{\mathsf{HORPO}} u'$

where:

- $\succ\!\!\succ^{\mathsf{HORPO}}$ is a relation between a term and a set of terms, defined as: $t = f(t_1, \ldots, t_k) \succ\!\!\succ^{\mathsf{HORPO}} \{u_1, \ldots, u_n\}$ iff $\forall_{i \in \{1,\ldots,n\}}\ t \succ^{\mathsf{HORPO}} u_i \vee (\exists_j\ t_j \succeq^{\mathsf{HORPO}} u_i)$,
- $\succeq^{\mathsf{HORPO}}$ is the reflexive closure of HORPO (that is $\succeq^{\mathsf{HORPO}} \equiv \succ^{\mathsf{HORPO}} \cup =$) and
- $\succ^{\mathsf{HORPO}}_{MUL}$ is a multiset extension of HORPO (see Definition 7.3). $\diamond$

Note that we will not prove transitivity of the ordering, neither in this presentation nor in Coq. Although this version of HORPO is transitive [JR99], many others are not [JR07], but as they are well-founded, their transitive closures are well-founded orderings. This justifies our abuse of terminology in calling HORPO an ordering. Thanks to defining the multiset extension in Section 7.2 for an arbitrary relation and not just for orderings, transitivity of the ordering plays no role in this development. It also allows for easier extension of the formalization for more complex, non-transitive versions of HORPO.

Let us now discuss the relation with the definition from [JR99]. There are three major differences. First let us note that in our variant only terms of equal types can be compared whereas in the original definition this restriction is weaker and it is possible to compare terms of equivalent types, where equivalence of types is a congruence generated by equating all sorts (in other words two types are equivalent if they have the same arrow structure).

The reason for strengthening this assumption is that allowing to reduce between different sorts poses some technical difficulties. In [JR99] this problem was solved by extending the typing rules with the congruence rule which presence is basically equivalent to collapsing all sorts and which allows typing terms that normally would be ill-typed due to a sort clash. Our goal was to use $\lambda^{\rightarrow}$ in its purest form as a meta-language and hence we decided not to do that.

But there is an even stronger argument against implementing this extension. It would complicate the formalization, whereas, as we claim, it would have absolutely no practical advantage. That is because the presented variant of HORPO takes no advantage of the sorting information. So we can prove some system terminating if and only if we can prove termination of its variant with all sorts collapsed to one single sort. Now, obviously, termination of such collapsed system implies termination of the original one. So it makes perfect sense to restrict (or transform) our termination problems to one-sorted setting.

The second difference is that the original definition of HORPO uses statuses and allows arguments of function symbols to be compared either lexicographically or as multisets, depending on the status (in the same way as for the first-order RPO; compare Section 1.3.4), whereas we allow only for comparing arguments of functions as multisets. This choice was made simply to avoid dealing with statuses and multiset comparison has been chosen as posing more difficulties (proofs of its well-foundedness and decidability are much more intricate than in the case of lexicographic order), so extension with statuses and lexicographic comparison should be relatively easy.

Finally we use the multiset extension as in Definition 7.3 instead of the one from Definition 7.2. Case (ii) of Lemma 7.8 will be crucial for the results in Section 9.3.2 and for the Definition 7.2 only its weaker variant holds (Lemma 7.8 (i)).

We conclude this section with a simple termination argument using HORPO.

Example 9.16. *Consider the one sorted variant of the higher-order term rewriting system from Example 9.6, with the following signature:*

$$\mathcal{F} = \{\mathrm{nil} : \star, \quad \mathrm{cons} : \star \to \star \to \star, \quad \mathrm{map} : \star \to (\star \to \star) \to \star\}$$

and its two rules:

$$
\begin{array}{rcrcl}
F : \star \to \star & \vdash & \mathrm{map}(\mathrm{nil}, F) & \to & \mathrm{nil} \\
x : \star, l : \star, F : \star \to \star & \vdash & \mathrm{map}(\mathrm{cons}(x, l), F) & \to & \mathrm{cons}(@(F, x), \mathrm{map}(l, F))
\end{array}
$$

We will orient the rules of this system using HORPO*. The first one is trivial by* (H_1)*. For the second one we take precedence with* $\mathrm{map} > \mathrm{cons}$ *and apply* (H_2)*. The remaining obligations are* $\mathrm{map}(\mathrm{cons}(x, l), F) \succ^{\mathsf{HORPO}} @(F, x)$ *and* $\mathrm{map}(\mathrm{cons}(x, l), F) \succ^{\mathsf{HORPO}} \mathrm{map}(l, F)$*. The latter is easily shown by* (H_3) *and* (H_1)*. The first is taken care of by* (H_5) *followed by two applications of* (H_1)*.* $\lhd$

Coq ▸ The Coq variant of the Definition 9.15 consists of five mutually recursive inductive definitions for: $\gg^{\mathsf{HORPO}}$, $\succeq^{\mathsf{HORPO}}$, $\succ^{\mathsf{HORPO}}_{MUL}$ and $\succ^{\mathsf{HORPO}}$, the last one split over two definitions (one factoring out the condition that terms under consideration must have equal types and environments). For details of the definition we refer to [Kop06b]. The Example 9.16 has also been carried out in Coq. ◂

9.3.2 Properties of HORPO

In this section we will prove some properties of HORPO.

Lemma 9.17. HORPO *is stable under substitution, that is:*

$$t \succ^{\mathsf{HORPO}} u \implies t\gamma \succ^{\mathsf{HORPO}} u\gamma$$

Proof. Induction on pair (t, u) ordered by $(\sqsubset, \sqsubset)_{lex}$ followed by a case analysis on $t \succ^{\mathsf{HORPO}} u$.

(H_1) $t = f(t_1, \ldots, t_n)$ and $t_i \succeq^{\mathsf{HORPO}} u$ for some $i \in \{1, \ldots, n\}$. But then $t\gamma = f(t_1\gamma, \ldots, t_n\gamma) \succ^{\mathsf{HORPO}} u\gamma$ by (H_1) since $t_i\gamma \succeq^{\mathsf{HORPO}} u\gamma$ by the induction hypothesis.

(H_2) $t = f(t_1, \ldots, t_n)$, $u = g(u_1, \ldots, u_k)$, $f \rhd g$ and $t \gg^{\mathsf{HORPO}} \{u_1, \ldots, u_k\}$. But then to get $t\gamma \succ^{\mathsf{HORPO}} u\gamma$ by (H_2) we only need to show $t\gamma \gg^{\mathsf{HORPO}} \{u_1\gamma, \ldots, u_k\gamma\}$. For every $i \in \{1, \ldots, k\}$ we have $t \succ^{\mathsf{HORPO}} u_i \vee (\exists_j\, t_j \succeq^{\mathsf{HORPO}} u_i)$. In either case we have $t\gamma \succ^{\mathsf{HORPO}} u_i\gamma$ or $t_j\gamma \succeq^{\mathsf{HORPO}} u_i\gamma$ by the induction hypothesis.

(H_3) $t = f(t_1, \ldots, t_n)$, $u = f(u_1, \ldots, u_k)$ and $\{\{t_1, \ldots t_n\}\} \succ^{\mathsf{HORPO}}_{MUL} \{\{u_1, \ldots, u_k\}\}$. But then we have $\{\{t_1\gamma, \ldots t_n\gamma\}\} \succ^{\mathsf{HORPO}}_{MUL} \{\{u_1\gamma, \ldots, u_k\gamma\}\}$ since for all $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, k\}$, $t_i \succ^{\mathsf{HORPO}} u_j$ implies $t_i\gamma \succ^{\mathsf{HORPO}} u_j\gamma$ by the induction hypothesis. So we get $t\gamma \succ^{\mathsf{HORPO}} u\gamma$ by (H_3).

(H_4) $@(u_1, \ldots, u_k)$ is a partial left-flattening of u and $t \gg^{\mathsf{HORPO}} \{u_1, \ldots u_k\}$. We use the same partial left-flattening for $u\gamma$ and get $t\gamma \gg^{\mathsf{HORPO}} \{u_1\gamma, \ldots, u_k\gamma\}$ with the same argument as in case (H_2). We conclude $t\gamma \succ^{\mathsf{HORPO}} u\gamma$ by (H_4).

(H_5) $t = @(t_l, u_r)$, $u = @(t_l, u_r)$ and $\{\{t_l, t_r\}\} \succ^{\mathsf{HORPO}}_{MUL} \{\{u_l, u_r\}\}$. Type considerations show that $t_l \succeq^{\mathsf{HORPO}} u_l$, $t_r \succeq^{\mathsf{HORPO}} u_r$ and $t_l \succ^{\mathsf{HORPO}} u_l \vee t_r \succ^{\mathsf{HORPO}} u_r$. By induction hypothesis on $(t_l\gamma, u_l\gamma)$ and $(t_r\gamma, u_r\gamma)$ we conclude $\{\{t_l\gamma, t_r\gamma\}\} \succ^{\mathsf{HORPO}}_{MUL} \{\{u_l\gamma, u_r\gamma\}\}$ and hence $t\gamma \succ^{\mathsf{HORPO}} u\gamma$ by (H_5).

(H_6) $t = \lambda x:\sigma.t'$, $u = \lambda x:\sigma.u'$ and $t' \succ^{\mathsf{HORPO}} u'$. But then $t\gamma = \lambda x:\sigma.t'\gamma$, $u\gamma = \lambda x:\sigma.u'\gamma$ and $t'\gamma \succ^{\mathsf{HORPO}} u'\gamma$ by the induction hypothesis. So $t\gamma \succ^{\mathsf{HORPO}} u\gamma$ by (H_6). □

Lemma 9.18. HORPO *is monotonous, that is:*

$$u \succ^{\mathsf{HORPO}} u' \implies t[u]_p \succ^{\mathsf{HORPO}} t[u']_p$$

Proof. The proof proceeds by induction on p and essentially uses the following observations:

- if $w_r \succ^{\mathsf{HORPO}} w_r'$ then $@(w_l, w_r) \succ^{\mathsf{HORPO}} @(w_l, w_r')$ by (H_5).
- if $w_l \succ^{\mathsf{HORPO}} w_l'$ then $@(w_l, w_r) \succ^{\mathsf{HORPO}} @(w_l', w_r)$ by (H_5).
- if $w \succ^{\mathsf{HORPO}} w'$ then $f(\ldots, w, \ldots) \succ^{\mathsf{HORPO}} f(\ldots, w', \ldots)$ by (H_3).
- if $w \succ^{\mathsf{HORPO}} w'$ then $\lambda x:\sigma.w \succ^{\mathsf{HORPO}} \lambda x:\sigma.w'$ by (H_6). □

Lemma 9.19. HORPO *preserves variables, that is:*

$$t \succ^{\mathsf{HORPO}} u \implies \mathrm{Vars}(t) \supseteq \mathrm{Vars}(u)$$

Proof. The proof uses the same inductive argument as in the above proof of stability of HORPO under substitution and all cases are easy. □

Lemma 9.20. HORPO *is compatible with* $\sim$*, that is:*

$$\left.\begin{array}{c} \Gamma \vdash t : \delta \succ^{\mathsf{HORPO}} \Delta \vdash u : \eta \\ \Gamma \vdash t : \delta \overset{Q}{\sim} \Gamma' \vdash t' : \delta' \\ \Delta \vdash u : \eta \overset{Q}{\sim} \Delta' \vdash u' : \eta' \\ \Gamma' = \Delta' \end{array}\right\} \implies \Gamma' \vdash t' : \delta' \succ^{\mathsf{HORPO}} \Delta' \vdash u' : \eta'$$

Proof. The proof is slightly technical but the inductive argument is again the same: induction on pair of terms (t, u) ordered by lexicographic extension of the subterm relation. □

Lemma 9.21. *If* $t \in \mathbb{C}$ *and* $t \succeq^{\mathsf{HORPO}} u$ *then* $u \in \mathbb{C}$*.*

Proof. We either have $t = u$, but then $u = t \in \mathbb{C}$, or $t \succ^{\mathsf{HORPO}} u$ in which case $u \in \mathbb{C}$ by the computability property (C_1). □

We conclude this section with a result that is not present in [JR99], namely a proof of the fact that $\succ^{\mathsf{HORPO}}$ is decidable.

Theorem 9.22. *Given terms t and u and a decidable precedence $\rhd$, the problem whether $t \succ^{\mathsf{HORPO}} u$ is decidable.*

Proof. Induction on the pair (t, u) ordered by $(\sqsubset, \sqsubset)_{lex}$ followed by a case analysis on t.

- $t = x$. Variables are in normal forms with respect to $\succ^{\mathsf{HORPO}}$ so we cannot have $x \succ^{\mathsf{HORPO}} u$.
- $t = @(t_l, t_r)$. Only (H_5) is applicable if $u = @(u_l, u_r)$ and for that, taking typing considerations into account, it is required that $t_l \succeq^{\mathsf{HORPO}} u_l$, $t_r \succeq^{\mathsf{HORPO}} u_r$ and $t_l \succ^{\mathsf{HORPO}} u_l \vee u_l \succ^{\mathsf{HORPO}} u_r$ all of which are decidable by the induction hypothesis.
- $t = \lambda x\!:\!\sigma.t_b$. Only (H_6) is applicable for $u = \lambda x\!:\!\sigma.u_b$ and it is required that $t_b \succ^{\mathsf{HORPO}} u_b$ which we can decide by induction hypothesis.
- $t = f(t_1, \ldots, t_n)$. We have several cases to consider corresponding to application of different clauses of HORPO:
  - (H_1): for every $i \in \{1, \ldots, n\}$ we check whether $t_i \succeq^{\mathsf{HORPO}} u$ by the application of the induction hypothesis.
  - (H_2): u needs to be of the shape $u = g(u_1, \ldots, u_k)$ with $f \rhd g$ (we assume precedence to be decidable). We need to check whether $t \succ\succ^{\mathsf{HORPO}} \{u_1, \ldots, u_k\}$. So for every $i \in \{1, \ldots, k\}$ we check whether $t \succ^{\mathsf{HORPO}} u_i$ or $t_j \succ^{\mathsf{HORPO}} u_i$ for some $j \in \{1, \ldots, n\}$. Typing considerations are helpful in immediate discarding of many cases.
  - (H_3): comparison between all arguments of t and u is decidable by the induction hypothesis so to conclude whether multisets of arguments can be compared we use Theorem 7.7.
  - (H_4): we consider all possible partial left-flattenings $@(u_1, \ldots, u_k)$ of u (bounded by the size of u) and for each of them we check whether $t \succ\succ^{\mathsf{HORPO}} \{u_1, \ldots, u_k\}$ in the same way as in the (H_2) case. $\square$

9.3.3 Well-foundedness of HORPO

In this section we present the proof of well-foundedness of $\succ^{\mathsf{HORPO}} \cup \rightarrow_\beta$. This relation will play an important role in this section so let us abbreviate it by $\rightsquigarrow \equiv \succ^{\mathsf{HORPO}} \cup \rightarrow_\beta$. For the proof we will use the computability predicate proof method due to Tait and Girard which was discussed in Section 9.2.

Property	**Proof for $\to_\beta$**	**Proof for $\succ^{\mathsf{HORPO}}$**
(P_1)	Theorem 8.44	Direct from the definition.
(P_2)	Direct from the definition	Direct from the definition.
(P_3)	Lemma 8.45	Lemma 9.19
(P_4)	Direct from the definition	Direct from the definition.
(P_5)	Lemma 8.46	Lemma 9.20
(P_6)	Lemma 8.47	Lemma 9.17
(P_7)	Lemma 8.49	Lemma 9.18
(P_8)	Direct from the definition	Direct from the definition
(P_9)	Direct from the definition	Direct from the definition.

Table 9.3 Conformance of $\to_\beta$ and $\succ^{\mathsf{HORPO}}$ to properties required by computability (summarized in Table 9.1)

Note that we will use computability with respect to $\rightsquigarrow$ and for that we need to prove properties (P_1)-(P_9) for $\rightsquigarrow$. In Table 9.3 conformance of $\to_\beta$ and $\succ^{\mathsf{HORPO}}$ to those properties is summarized. Note that all those properties easily generalize to the union if they hold for the components.

The crucial lemma states that if function arguments are computable then so is the function application. First we need an auxiliary lemma.

Lemma 9.23. *For any $t = f(t_1, \ldots, t_n)$ and $u = g(u_1, \ldots, u_k)$ if*

(1) $t \succ\!\!\succ^{\mathsf{HORPO}} \{u_1, \ldots u_k\}$ and

(2) $\forall_{i \in \{1,\ldots,n\}}\ t_i \in \mathbb{C}$ and

(3) $\forall_{j \in \{1,\ldots,k\}}\ t \rightsquigarrow u_j \implies u_j \in \mathbb{C}$,

then $\forall_{j \in \{1,\ldots,k\}}\ u_j \in \mathbb{C}$.

Proof. For a given u_j according to the definition of $\succ\!\!\succ^{\mathsf{HORPO}}$ we have two cases:

- $t \succ^{\mathsf{HORPO}} u_j$, then $u_j \in \mathbb{C}$ by assumption (3).
- $t_i \succeq^{\mathsf{HORPO}} u_j$ for some i. If $t_i = u_j$ then $t_j \in \mathbb{C}$ by assumption (2) and so $t_i \in \mathbb{C}$. Otherwise $t_i \succ^{\mathsf{HORPO}} u_j$ but $t_i \in \mathbb{C}$ by (2) and then $u_j \in \mathbb{C}$ by (C_1). □

Now we can present the aforementioned lemma.

Lemma 9.24. *If $t_1, \ldots, t_n \in \mathbb{C}$ then $t = f(t_1, \ldots, t_n) \in \mathbb{C}$.*

Proof. The proof proceeds by well-founded induction on the pair of a function symbol and a multiset of computable terms, $(f, \{\{t_1, \ldots, t_n\}\})$, ordered lexicographically by $(\rhd, \rightsquigarrow_{mul})_{lex}$. Note that all terms in the multiset are computable and hence strongly normalizable, by (C_2). So $(\rhd, \rightsquigarrow_{mul})_{lex}$ is well-founded by Theorem 7.20 and Theorem 1.6 which justifies the induction argument.

Since t is neutral we apply (C_3) and we are left to show that $u \in \mathbb{C}$ for an arbitrary u, such that $t \rightsquigarrow u$. We will show that by inner induction on the structure of u. We continue by case analysis on $t \rightsquigarrow u$. The first case corresponds to a beta-reduction step and the following ones to applications of clauses (H_1), (H_2), (H_3) and (H_4) of HORPO definition. Note that clauses (H_5) and (H_6) are not applicable.

(β) Let $t \rightarrow_\beta u$. By Lemma 8.48 we know that the reduction is in one of the arguments, so for some j we have $u = f(t_1, \ldots t'_j, \ldots t_n)$ with $t_j \rightarrow_\beta t'_j$. For every i, $t_i \in \mathbb{C}$ by assumption and $t'_j \in \mathbb{C}$ by (C_1) so we conclude $u \in \mathbb{C}$ by the outer induction hypothesis.

(H_1) $t_i \succeq^{\mathsf{HORPO}} u$ for some $i \in \{1, \ldots, n\}$. By assumption $t_i \in \mathbb{C}$ and we either have $t_i = u$ or $t_i \succ^{\mathsf{HORPO}} u$ but then $u \in \mathbb{C}$ by (C_1).

(H_2) $u = g(u_1, \ldots u_k)$ with $f \rhd g$. All $u_i \in \mathbb{C}$ for $i \in \{1, \ldots, k\}$ by Lemma 9.23 and we conclude that $u \in \mathbb{C}$ by the outer induction hypothesis as $(f, \{\{t_1, \ldots t_n\}\})$ $(\rhd, \rightsquigarrow_{mul})_{lex}$ $(g, \{\{u_1, \ldots, u_k\}\})$

(H_3) $u = f(u_1, \ldots u_k)$ with $\{\{t_1, \ldots, t_n\}\} \succ^{\mathsf{HORPO}}_{MUL} \{\{u_1, \ldots, u_k\}\}$. We can conclude $u \in \mathbb{C}$ by the outer induction hypothesis if we can prove that $u_i \in \mathbb{C}$ for $i \in \{1, \ldots, k\}$. For arbitrary i, by Lemma 7.8 (ii) we get $t_j \succeq^{\mathsf{HORPO}} u_i$ for some j and since $t_j \in \mathbb{C}$ by assumption we conclude $u_i \in \mathbb{C}$ by (C_1).

(H_4) $@(u_1, \ldots, u_k)$ is some partial left-flattening of u and $t \ggcurly^{\mathsf{HORPO}} \{u_1, \ldots, u_k\}$. By Lemma 9.23 we get $u_i \in \mathbb{C}$ for $i \in \{1, \ldots, k\}$ and hence $u \in \mathbb{C}$. ☐

The next step is to show that application of a computable substitution gives computable term, where we define computable substitution as a substitution containing in its domain only computable terms. More formally:

Definition 9.25. [Computable substitution] We say that a substitution $\gamma = [x_1/u_1, \ldots, x_n/u_n]$ is a *computable substitution* if for every $i \in \{1, \ldots, n\}$, $u_i \in \mathbb{C}$. ◇

Lemma 9.26. *Let γ be a computable substitution and t be an arbitrary term. Then $t\gamma \in \mathbb{C}$.*

Proof. We proceed by induction on the structure of t. We have the following cases to consider.

- $t = x$. If $x \in Dom(\gamma)$ then $\gamma = [\ldots, x/u, \ldots]$ and $t\gamma = u$ but $u \in \mathbb{C}$ since γ is a computable substitution. Otherwise $x \notin Dom(\gamma)$ and $t\gamma = x \in \mathbb{C}$ by (C_4).
- $t = f(t_1, \ldots, t_n)$ so $t\gamma = f(t_1\gamma, \ldots, t_n\gamma)$. We apply Lemma 9.24 and we are left to show that for $i \in \{1, \ldots, n\}$, $t_i \in \mathbb{C}$ which easily follows from the induction hypothesis.
- $t = @(t_l, t_r)$ and $t\gamma = @(t_l\gamma, t_r\gamma)$. Both $t_l\gamma$ and $t_r\gamma$ are computable by the induction hypothesis so $t\gamma \in \mathbb{C}$ by the definition of computability.
- $t = \lambda x : \sigma . t_b$ so $t\gamma = \lambda x : \sigma . t_b\gamma$. By application of (C_5) we are left to show that $t_b\gamma[x/u] \in \mathbb{C}$ for any $u \in \mathbb{C}_\sigma$. But $t_b\gamma[x/u] = t_b(\gamma \cup [x/u])$ since $x \notin Dom(\gamma)$. Since $\gamma \cup [x/u]$ is a computable substitution as so is γ and $u \in \mathbb{C}$, we can conclude $t\gamma \in \mathbb{C}$ by the induction hypothesis. $\square$

Now we are ready to present the main theorem stating that the union of HORPO and β-reduction of simply typed λ-calculus, is a well-founded relation on terms.

Theorem 9.27. *The relation $\rightsquigarrow$ is well-founded.*

Proof. We need to show that $t \in \mathcal{A}cc$ for an arbitrary t. Consider the empty substitution ϵ, which is computable by definition. We also have $t = t\epsilon$ so we conclude $t \in \mathbb{C}$ by Lemma 9.26 and then $t \in \mathcal{A}cc$ by (C_2). $\square$

In this way we we have essentially proven that $\rightsquigarrow$ is a, so-called, *higher-order reduction ordering* [JR06] (the counter-part of the first-order reduction ordering; see Definition 1.20) as (we refer to the article for the definitions of respective notions):

- *well-foundedness* was proven in Theorem 9.27.
- *coherence* follows from compatibility with $\sim$, Lemma 8.46 and Lemma 9.20,
- *stability* was proven in Lemma 8.47 and Lemma 9.17,
- *monotonicity* was proven in Lemma 8.49 and Lemma 9.18,
- *functionality* states that $\rightarrow_\beta$ is included in the ordering and indeed we defined $\rightsquigarrow$ as $\succ^{\mathsf{HORPO}} \cup \rightarrow_\beta$.

Conclusions

In this thesis we presented contributions to methods for producing proofs of termination of term rewriting systems and to certification of such proofs.

The contributions with regard to proving termination are:

1. extending the arctic interpretations methods from string to term rewriting and generalizing from arctic naturals to arctic integers (Section 2.2),
2. direct approach for applying RPO on infinite systems obtained via application of semantic labeling with natural numbers (Section 3.1),
3. an improved approach to the above where semantic labeling is replaced with predictive labeling, the search procedure is realized via an encoding into SAT and the whole approach is implemented within the dependency pair setting (Section 3.2) and
4. application of termination proving techniques to proving liveness properties of systems (Chapter 5).

Our efforts in the area of certification can be summarized by the following contributions to the CoLoR project:

1. formalization and extension of CoLoR with the ability to formalize termination proofs using the matrix interpretations method (Section 2.3.4),
2. ditto for two variants of arctic interpretations (Section 2.3.5),
3. a termination prover implementing all the techniques available in CoLoR and allowing to produce formal certificates along with generated termination proofs (Chapter 4) and
4. formalization of the higher-order recursive path ordering (HORPO), an extension of the well-known termination technique of recursive path ordering (RPO) to the higher-order setting (Part II of the thesis).

A lot remains to be done in this area. We mentioned some possible extensions and directions for further work at the end of each chapter. However, we believe that the main goal is to extend the applicability of CoLoR, on the one hand by increasing the number of techniques available for certification and, on the other hand, by improving the flexibility of application of those methods, so that all the variants in which those methods are used by the tools would be covered. The ultimate objective should be the ability to certify a major part of the termination proofs produced by modern termination provers.

References

[ABF$^+$05] B. E. Aydemir, A. Bohannon, M. Fairbairn, J. N. Foster, B. C. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, and S. Zdancewic. Mechanized metatheory for the masses: The POPLmark challenge. In *Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics (TPHOLs '05)*, volume 3603 of *Lecture Notes in Computer Science*, pages 50–65, 2005. 119

[ADHS01] T. Altenkirch, P. Dybjer, M. Hofmann, and P. J. Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *Proceedings of the 16th annual IEEE Symposium on Logic in Computer Science (LICS '01)*, pages 303–310, 2001. 119

[AG00] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000. 25, 26, 32, 60, 89, 91

[Alt93a] T. Altenkirch. A formalization of the strong normalization proof for system F in LEGO. In *Proceedings of the 1st International Conference on Typed Lambda Calculi and Applications (TLCA '93)*, volume 664 of *Lecture Notes in Computer Science*, pages 13–28, 1993. 119

[Alt93b] T. Altenkirch. Proving strong normalization of CC by modifying realizability semantics. In *Proceedings of the International Workshop on Types for Proofs and Programs (TYPES '93)*, volume 806 of *Lecture Notes in Computer Science*, pages 3–18, 1993. 119

[Ama05] R. M. Amadio. Synthesis of max-plus quasi-interpretations. *Fundamenta Informaticae*, 65(1-2):29–60, 2005. 48

[Bar92] H. P. Barendregt. Lambda calculi with types. *Handbook of logic in computer science (vol. II)*, pages 117–309, 1992. 131

[Bar99] B. Barras. *Auto-validation d'un système de preuves avec familles inductives.* PhD thesis, Université Paris 7, Paris, France, November 1999. 119

[BBLS06] U. Berger, S. Berghofer, P. Letouzey, and H. Schwichtenberg. Program extraction from normalization proofs. *Studia Logica*, 82:25–49, 2006. 119

[BC04] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer Verlag, 2004. 28, 115

[BDCG+06] F. Blanqui, W. Delobel, S. Coupet-Grimal, S. Hinderer, and A. Koprowski. CoLoR, a Coq library on rewriting and termination. In *8th International Workshop on Termination (WST '06)*, pages 69–73, 2006. 116, 117

[Ber04] C. Berghofer. A constructive proof of Higman's lemma in Isabelle. In *Proceedings of the International Workshop on Types for Proofs and Programs (TYPES '03)*, volume 3085 of *Lecture Notes in Computer Science*, pages 66–82, 2004. 120

[BKN07] L. Bulwahn, A. Krauss, and T. Nipkow. Finding lexicographic orders for termination proofs in Isabelle/HOL. In *Proceedings of the 20th International Conference on Theorem Proving in Higher Order Logics (TPHOLs '07)*, volume 4732 of *Lecture Notes in Computer Science*, pages 38–53, 2007. 29

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. 13, 124

[Bru72] N. G. de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indag. Math.*, 34(5):381–392, 1972. 133, 161

[CCF+07] E. Contejean, P. Courtieu, J. Forest, O. Pons, and X. Urbain. Certification of automated termination proofs. In *Proceedings of the 6th International Symposium on Frontiers of Combining Systems (FroCoS '07)*, volume 4720 of *Lecture Notes in Computer Science*, pages 148–162, 2007. 29, 91

[CGD06] S. Coupet-Grimal and W. Delobel. An effective proof of the well-foundedness of the multiset path ordering. *Applicable Algebra in Engineering, Communication and Computing*, 17(6):453–469, 2006. 120

[CHP71] P. J. Courtois, F. Heymans, and D. L. Parnas. Concurrent control with "readers" and "writers". *Communications of the ACM*, 14(10):667–668, 1971. 109

[CHR92] P.-L. Curien, T. Hardin, and A. Ríos. Strong normalization of substitutions. In *Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science (MFCS '92)*, volume 629 of *Lecture Notes in Computer Science*, pages 209–217, 1992. 60

[Chu40] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, 1940. 131

[CL87] A. B. Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–159, 1987. 63

[CMMU] E. Contejean, C. Marché, B. Monate, and X. Urbain. The CiME rewrite tool. `http://cime.lri.fr/`. 91

[COL] CoLoR: a Coq library on rewriting and termination. `http://color.loria.fr/`. 28, 116, 117

[COQ] The Coq proof assistant. `http://coq.inria.fr/`. 28

[CSKL+06] M. Codish, P. Schneider-Kamp, V. Lagoon, R. Thiemann, and J. Giesl. SAT solving for argument filterings. In *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '06)*, volume 4246 of *Lecture Notes in Computer Science*, pages 30–44, 2006. 78, 81, 82

[Der82] N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982. 20, 21, 89, 115

[End05] J. Endrullis. Jambox. `http://joerg.endrullis.de/`, 2005. 41, 59

[ES03] N. Eén and N Sörensson. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518, 2003. 46, 81

[EWZ08] J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning*, 40(2–3):195–220, 2008. 6, 17, 18, 31, 32, 36, 46, 56, 78, 79, 82, 89

[FGM+08] C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. Maximal termination. In *Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA '08)*, volume 5117 of *Lecture Notes in Computer Science*, pages 110–125, 2008. 63, 72, 84

[Fra86] N. Francez. *Fairness*. Springer-Verlag, 1986. 93

[Fuc62] L. Fuchs. *Partially Ordered Algebraic Systems*. Addison-Wesley, 1962. 35

[FZ95] M. C. F. Ferreira and H. Zantema. Dummy elimination: Making termination easier. In *Proceedings of the 10th International Symposium on Fundamentals of Computation Theory (FCT '95)*, volume 965 of *Lecture Notes in Computer Science*, pages 243–252, 1995. 89

[GAO02] J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002. 47

[Gie95] J. Giesl. Termination analysis for functional programs using term orderings. In *Proceedings of the 2nd International Symposium on Static Analysis (SAS '95)*, volume 983 of *Lecture Notes in Computer Science*, pages 154–171, 1995. 5

[Gol99] J. S. Golan. *Semirings and their Applications*. Kluwer, 1999. 35

[GP97] S. Gaubert and M. Plus. Methods and applications of (MAX, +) linear algebra. In *Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science (STACS '97)*, volume 1200 of *Lecture Notes in Computer Science*, pages 261–282, 1997. 35

[Gra94] B. Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 5:131–158, 1994. 85

[GSSKT06] J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated termination analysis for haskell: From term rewriting to programming languages. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *Lecture Notes in Computer Science*, pages 297–312, 2006. 5

[GTL89] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and types"*, volume 7 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 1989. 116, 151, 154

[GTSK04] J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '04)*, volume 3452 of *Lecture Notes in Computer Science*, pages 301–331, 2004. 25, 26, 57, 60

[GTSKF04] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA '04)*, volume 3091 of *Lecture Notes in Computer Science*, pages 210–220, 2004. 43, 59

[GTSKF06] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006. 25, 26, 27, 76

[GTSSK07] J. Giesl, R. Thiemann, S. Swiderski, and P. Schneider-Kamp. Proving termination by bounded increase. In *Proceedings of the 21st International Conference on Automated Deduction (CADE '07)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 443–459, 2007. 32, 45, 56

[GZ03a] J. Giesl and H. Zantema. Liveness in rewriting. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications (RTA '03)*, volume 2706 of *Lecture Notes in Computer Science*, pages 321–336, 2003. 7, 88, 93, 94, 95, 96, 97, 98, 101, 107, 109

[GZ03b] J. Giesl and H. Zantema. Simulating liveness by reduction strategies. In *Proceedings of the Workshop on Reduction Strategies in Rewriting and Programming (WRS '03)*, volume 86 of *Electronic Notes in Theoretical Computer Science*, 2003. 93

[Hin97] J. R. Hindley. *Basic Simple Type Theory*. Cambridge University Press, 1997. 136

[Hin04] S. Hinderer. Certification of termination proofs based on polynomial interpretations. Master's thesis, Université Henri Poincaré, Nancy, France, June 2004. 32, 53

[HJ98] H. Hong and D. Jakuš. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21(1):23–38, 1998. 19, 39, 63

[HL86] T. Hardin and A. Laville. Proof of termination of the rewriting system SUBST on CCL. *Theoretical Computer Science*, 46(2–3):305–312, 1986. 60

[HM04] N. Hirokawa and A. Middeldorp. Polynomial interpretations with negative coefficients. In *Proceedings of the 7th International Conference on Artificial Intelligence and Symbolic Computation (AISC '04)*, volume 3249 of *Lecture Notes in Computer Science*, pages 185–198, 2004. 32, 43

[HM05] N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005. 77

[HM06] N. Hirokawa and A. Middeldorp. Predictive labeling. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *Lecture Notes in Computer Science*, pages 313–327, 2006. 24, 25, 60, 75, 76

[HM07] N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007. 25, 77

[Hoa74] C. A. R. Hoare. Monitors: an operating system structuring concept. *Communications of the ACM*, 17(10):549–557, 1974. 109

[HW06] D. Hofbauer and J. Waldmann. Termination of string rewriting with matrix interpretations. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *Lecture Notes in Computer Science*, pages 328–342, 2006. 31, 33, 34

[JO91] J.-P. Jouannaud and M. Okada. A computation model for executable higher-order algebraic specification languages. In *Proceedings of the 6th annual IEEE Symposium on Logic in Computer Science (LICS '91)*, pages 350–361, 1991. 115, 118, 153, 162

[JR99] J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proceedings of the 14th annual IEEE Symposium on Logic in Computer Science (LICS '99)*, pages 402–411, 1999. 7, 115, 116, 117, 118, 120, 127, 154, 155, 162, 163, 166

[JR01] J.-P. Jouannaud and A. Rubio. Higher-order recursive path orderings 'à la carte'. http://www.lix.polytechnique.fr/Labo/Jean-Pierre.Jouannaud/biblio.html, 2001. 115, 116, 143, 153, 154

[JR06] J.-P. Jouannaud and A. Rubio. Higher-order orderings for normal rewriting. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *Lecture Notes in Computer Science*, pages 387–399, 2006. 115, 169

[JR07] J.-P. Jouannaud and A. Rubio. Polymorphic higher-order recursive path orderings. *Journal of the ACM*, 54(1):1–48, 2007. 115, 136, 162

[Kle03] N. de Kleijn. Well-foundedness of RPO in Coq. Master's thesis, Vrije Universiteit, Amsterdam, The Netherlands, August 2003. 120

[KM07] A. Koprowski and A. Middeldorp. Predictive labeling with dependency pairs using SAT. In *Proceedings of the 21st International Conference on Automated Deduction (CADE '07)*, volume 4603 of *Lecture*

Notes in Artificial Intelligence, pages 410–425, 2007. 24, 59, 60, 61, 91

[Kop04] A. Koprowski. Well-foundedness of the higher-order recursive path ordering in Coq. Technical Report TI-IR-004, Vrije Universiteit, Amsterdam, The Netherlands, August 2004. Master's thesis. 118, 129, 161

[Kop06a] A. Koprowski. Certified higher-order recursive path ordering. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *Lecture Notes in Computer Science*, pages 227–241, 2006. 119

[Kop06b] A. Koprowski. Coq formalization of the higher-order recursive path ordering. Technical Report CSR-06-21, Eindhoven University of Technology, Eindhoven, The Netherlands, August 2006. 116, 119, 122, 123, 125, 126, 127, 131, 136, 164

[Kop06c] A. Koprowski. TPA: Termination proved automatically. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA '06)*, volume 4098 of *Lecture Notes in Computer Science*, pages 257–266, 2006. 15, 29

[Kra07] A. Krauss. Certified size-change termination. In *Proceedings of the 21st International Conference on Automated Deduction (CADE '07)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 460–475, 2007. 29

[Kro92] D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP '92)*, volume 623 of *Lecture Notes in Computer Science*, pages 101–112, 1992. 48

[KSZM] M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. The Tyrolean Termination Tool 2 (TTT2). `http://colo6-c703.uibk.ac.at/ttt2`. 43, 91

[KW08] A. Koprowski and J. Waldmann. Arctic termination ... below zero. In *Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA '08)*, volume 5117 of *Lecture Notes in Computer Science*, pages 202–216, 2008. 31, 32

[KZ05] A. Koprowski and H. Zantema. Proving liveness with fairness using rewriting. Technical Report CS-Report 05-06, Tech. Univ. of Eindhoven, March 2005. 109, 111

[KZ06] A. Koprowski and H. Zantema. Automation of recursive path ordering for infinite labelled rewrite systems. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR '06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 332–346, 2006. 59, 60, 61, 78

[KZ08] A. Koprowski and H. Zantema. Certification of proving termination of term rewriting by matrix interpretations. In *Proceedings of the 34th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '08)*, volume 4910 of *Lecture Notes in Computer Science*, pages 328–339, 2008. 32, 55

[Lan79] D. S. Lankford. On proving term rewriting systems are noetherian. Technical Report MTP-3, Math. Dept., Louisiana Tech. Univ., Ruston, May 1979. 16, 18, 79, 89

[Lec95] F. Leclerc. Termination proof of term rewriting systems with the multiset path ordering: A complete development in the system Coq. In *Proceedings of the 2nd International Conference on Typed Lambda Calculi and Applications (TLCA '95)*, volume 902 of *Lecture Notes in Computer Science*, pages 312–327, 1995. 120

[LM08] S. Lucas and J. Meseguer. Termination of just/fair computations in term rewriting. *Information and Computation*, 206(5):652–675, 2008. 88

[Mag03] N. Magaud. Ring properties for square matrices, 2003. Coq contributions, available at `http://coq.inria.fr/contribs-eng.html`. 51

[Mur90] C. Murthy. *Extracting constructive content from classical proofs*. PhD thesis, Cornell University, New York, USA, 1990. 120

[Nip91] T. Nipkow. Higher-order critical pairs. In *Proceedings of the 6th annual IEEE Symposium on Logic in Computer Science (LICS '91)*, pages 342–349, 1991. 153

[Nip98] T. Nipkow. An inductive proof of the wellfoundedness of the multiset order. `http://www4.informatik.tu-muenchen.de/~nipkow/misc/index.html`, 1998. A proof due to W. Buchholz. 128, 130

[OCA] The Objective Caml language. `http://www.ocaml.org/`. 89

[Ohl94] E. Ohlebusch. On the modularity of termination of term rewriting systems. *Theoretical Computer Science*, 136(2):333–360, 1994. 85

[Per99] H. Persson. *Type Theory and the Integrated Logic of Programs*. PhD thesis, Göteborg University, Göteborg, Sweden, May 1999. 120

[PM93] C. Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In *Proceedings of the 1st International Conference on Typed Lambda Calculi and Applications (TLCA '93)*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345, 1993. 28

[Raa03] F. van Raamsdonk. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in TCS*, chapter 11, pages 588–668. Cambridge University Press, 2003. 153

[Rao88] J.-C. Raoult. Proving open properties by induction. *Information Processing Letters*, 29:19–23, 1988. 120

[RB86] M. Raynal and D. Beeson. *Algorithms for mutual exclusion*. MIT Press, 1986. 109

[RTA] The RTA list of open problems. `http://www.lsv.ens-cachan.fr/rtaloop`. 56

[SDB06] K. Stoevring, O. Danvy, and M. Biernacka. Program extraction from proofs of weak head normalization. In *Proceedings of the 21st Conference on the Mathematical Foundations of Programming Semantics (MFPS '06)*, volume 155 of *Electronic Notes in Theoretical Computer Science*, 2006. 119

[SGG04] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating system concepts*. John Wiley & Sons, Inc., 2004. 109

[SKGST06] P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination analysis for logic programs by term rewriting. In *Proceedings of the 16th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR '06)*, volume 4407 of *Lecture Notes in Computer Science*, pages 177–193, 2006. 5

[SKGST08] P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination proofs for logic programs by term rewiriting. *The Computing Research Repository*, abs/0803.0014, 2008. 5

[Tai67] W. W. Tait. Intentional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967. 7, 116, 151, 154

[TC] Termination competition. `http://www.lri.fr/~marche/termination-competition`. 15, 29, 31

[TeR03] TeReSe. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in TCS*. Cambridge University Press, 2003. 5, 13

[TG05] R. Thiemann and J. Giesl. The size-change principle and dependency pairs for termination of term rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 16(4):229–270, 2005. 24

[The04] The Coq development team. The Coq proof assistant reference manual, version 8.0. http://pauillac.inria.fr/coq/doc-eng.html, 2004. 115

[TM07] R. Thiemann and A. Middeldorp. Innermost termination of rewriting systems by labeling. In *Proceedings of the Workshop on Reduction Strategies in Rewriting and Programming (WRS '07)*, volume 204 of *Electronic Notes in Theoretical Computer Science*, 2007. 24

[TPD] Termination problem data base. www.lri.fr/~marche/tpdb. 15, 24, 44, 56, 81

[TZGSK08] R. Thiemann, H. Zantema, J. Giesl, and P. Schneider-Kamp. Adding constants to string rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 19(1):27–38, 2008. 40

[Urb04] X. Urbain. Modular & incremental automated termination proofs. *Journal of Automated Reasoning*, 32:315–355, 2004. 77

[Wal04] J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA '04)*, volume 3091 of *Lecture Notes in Computer Science*, pages 85–94, 2004. http://dfa.imn.htwk-leipzig.de/matchbox. 31, 46

[Wal07a] J. Waldmann. Arctic termination. In *9th International Workshop on Termination (WST '07)*, pages 1–4, 2007. 6, 31, 40

[Wal07b] J. Waldmann. Weighted automata for proving termination of string rewriting. *Journal of Automata, Languages and Combinatorics*, 12(4):545–570, 2007. 31

[WSc] Workshop on certified termination (wsct). http://termination-portal.org/wiki/WScT. 89

[Zan95] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24(1/2):89–105, 1995. 21, 22, 59, 60, 89

[Zan03] H. Zantema. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in TCS*, chapter 6, pages 181–259. Cambridge University Press, 2003. 21, 59, 60, 72

[Zan04] H. Zantema. TORPA: Termination of rewriting proved automatically. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA '04)*, volume 3091 of *Lecture Notes in Computer Science*, pages 95–104, 2004. 59

[Zan05] H. Zantema. Reducing right-hand sides for termination. In *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, pages 173–197, 2005. 89

[ZHM07] H. Zankl, N. Hirokawa, and A. Middeldorp. Constraints for argument filterings. In *Proceedings of the 33th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '07)*, volume 4362 of *Lecture Notes in Computer Science*, pages 579–590, 2007. 78

Index

Termination of Rewriting and Its Certification

Summary

In programming, *termination* of a program/algorithm means that its evaluation will eventually terminate, regardless of the input it receives. It is an important property and is required for *total correctness*. In general the problem is undecidable.

Term rewriting is a formal way of specifying computation and as such it can be seen as a generic model for programming languages. Termination, here meaning lack of infinite sequences, is a well-studied concept in this context. There exist a number of methods for proving termination as well as a number of tools for doing that automatically. There is an on-going work on application of this methodology and tools to proving termination of programs in actual programming languages.

In this thesis we first give a short introduction to term rewriting and to termination of rewriting. Subsequently we present a number of contributions to this field, which can be categorized into the following categories:

- proposing new methods for proving termination and refining the existing ones,
- developing a tool for proving termination and
- proposing a methodology and tools for *certification* of termination proofs, *i.e.*, formal verification of proofs produced by the existing tools for proving termination.

Samenvatting

Van een programma of algoritme betekent *terminatie* dat het uitvoeren ervan altijd zal eindigen, onafhankelijk van de betreffende invoer. Dit is een belangrijk begrip; zo is dit een van de eisen van *totale correctheid*. In zijn algemeenheid is het vaststellen van terminatie een onbeslisbaar probleem.

Termherschrijven is een formalisme om berekening te specificeren, en als zodanig kan het opgevat worden als generiek model voor programmeertalen. Hierin betekent terminatie het ontbreken van oneindig doorgaande berekeningen, een concept dat uitgebreid is bestudeerd. Zo is er een uitgebreid scala aan methoden om terminatie van termherschrijfsystemen te bewijzen, en zijn er diverse implementaties om dit geheel automatisch te doen. Huidig onderzoek richt zich onder meer op het toepassen van deze technieken op het bewijzen van terminatie van programma's in gebruikelijke programmeertalen.

In dit proefschrift geven we eerst een korte inleiding in herschrijven en terminatie van herschrijven. Vervolgens presenteren we enkele nieuwe bijdragen aan dit gebied, ingedeeld in de volgende onderdelen:

- het ontwikkelen van nieuwe methoden om terminatie te bewijzen en het verfijnen van bestaande methoden,
- het ontwkkelen van een tool om automatisch terminatie te bewijzen, en
- het ontwikkelen van methodiek en een tool om terminatiebewijzen te *certificeren*, dat wil zeggen het formeel verifiëren van de correctheid van bewijzen zoals die door bestaande tools worden opgeleverd.

Curriculum Vitae

Adam Koprowski was born on the 27th of March 1980 in Warsaw, Poland. In 1999, he graduated from the Czacki High School in Computer Science. He continued studying this subject at the Warsaw University. During his final year he received a scholarship and spent one year at the Vrije University Amsterdam (VU). In 2004 he graduated *cum laude* receiving his double Master's degree from the aforementioned institutions with the thesis entitled "Well-foundedness of the Higher Order Recursive Path Ordering in Coq".

Since September 2004, he has been a Ph.D. student within the Computer Science Department of the Eindhoven University of Technology (TU/e) in Eindhoven, The Netherlands. His research was funded by NWO (The Dutch Organization for Scientific Research) within the project "Verification and Rewriting". His research has led, apart from this thesis, to several publications at international conferences and in journals.

Titles in the IPA Dissertation Series since 2002

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments*. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications.* Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems.* Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs.* Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell.* Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation.* Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets.* Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics.* Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station.* Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents.* Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures.* Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling.* Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support- .* Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue.* Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of

Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space*. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization*. Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML*. Faculty of

Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models*. Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications*. Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition*. Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems*. Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains*. Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use*. Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems*. Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML*. Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery*. Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J. Mathijssen. *Logical Calculi for Reasoning with Binding*. Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks*. Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data*. Faculty of Mathematics and Computer Science, TU/e. 2007-19

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates*. Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathematics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calam. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems*. Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification*. Faculty of Mathematics and Computer Science, TU/e. 2008-24

Theorem 2:

Let $\mathcal{R}, \mathcal{R}'$ be TRSs over a

Let $(A, [\cdot], >, \gtrsim)$ be an extende

monotone Σ-algebra s.t.:

·) $\forall \alpha \; [l, \alpha] \gtrsim [r, \alpha]$ for every ru

·) $\forall \alpha \; [l, \alpha] > [r, \alpha]$ for every ru

Then $SN(\rightarrow_{\mathcal{R}})$ implies SN

Termination of Rewriting and its Certification

Termination of Rewriting and its Certification